UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Pedro de Carvalho Braga Ilídio Silva

On bipartite decision forests

São Carlos

2024

Pedro de Carvalho Braga Ilídio Silva

On bipartite decision forests

Dissertation presented to the Graduate Program in Physics at the Instituto de Física de São Carlos da Universidade de São Paulo, to obtain the degree of Master in Science.

Concentration area: Biomolecular Physics

Advisor: Prof. Dr. Otavio Henrique Thiemann

Coadvisor: Prof. Dr. Ricardo Cerri

Original version

São Carlos
2024

I AUTHORIZE THE REPRODUCTION AND DISSEMINATION OF TOTAL OR PARTIAL COPIES OF THIS DOCUMENT, BY CONVENTIONAL OR ELECTRONIC MEDIA FOR STUDY OR RESEARCH PURPOSE, SINCE IT IS REFERENCED.

Silva, Pedro de Carvalho Braga Ilídio
On bipartite decision forests / Pedro de Carvalho Braga Ilídio Silva; advisor Otavio Henrique Thiemann; co-advisor Ricardo Cerri -- São Carlos 2024.
123 p.

Dissertation (Master's degree - Graduate Program in Biomolecular Physics) -- Instituto de Física de São Carlos, Universidade de São Paulo - Brasil , 2024.

1. Decision forests. 2. Interaction prediction. 3. Bipartite learning. 4. Positive-unlabeled learning. I. Thiemann, Otavio Henrique, advisor. II. Cerri, Ricardo, co-advisor. III. Title.

FOLHA DE APROVAÇÃO

Pedro de Carvalho Braga Ilidio Silva

Dissertação apresentada ao Instituto de Física de São Carlos da Universidade de São Paulo para obtenção do título de Mestre em Ciências. Área de Concentração: Física Biomolecular.

Aprovado (a) em: 25/04/2024

Comissão Julgadora

Dr(a).: Ricardo Cerri

Instituição: (IFSC/USP)

Dr(a).: Gisele Lobo Pappa

Instituição: (UFMG/Belo Horizonte)

Dr(a).: Enrico Bertuzzo

Instituição: Diego Furtado Silva (ICMC/USP)

AGRADECIMENTOS

À Universidade de São Paulo e ao Instituto de Física de São Carlos (IFSC), por proverem os meios e os ambientes pelos quais este estudo pôde se desenvolver.

À Neusa, do Serviço de Biblioteca e Informação do IFSC, pelo atencioso auxílio com a normalização do presente documento.

Aos estimados orientadores Prof. Dr. Otavio Henrique Thiemann e Prof. Dr. Ricardo Cerri, pelas numerosas ideias, inspiração e direcionamento. Pelo acolhimento frente às repentinas readaptações acadêmicas que se fizeram necessárias. Pela confiança e compreensão com os atrasos, com os afluentes e com os contratempos durante todo o processo.

Ao André Alves, pelas várias discussões e colaborações enriquecedoras das quais grande parte desse trabalho teve origem.

Ao Prof. Dr. Ricardo de Marco, em memória, pelos muitos ensinamentos nos poucos anos que pudemos conviver.

Aos irmãos acadêmicos João Paulo, Luíza e Renan, pelo amparo e companhia essenciais durante os percalços do luto, da pandemia e das incertezas da vida.

Aos profissionais da saúde mental Marina e Pedro Paulo, pelo imprescindível apoio.

Aos prezados coabitantes Kauê e Roberto, pelo convívio agradável e encorajamento diário. Por ouvirem sempre gentilmente os desabafos e aflições.

Às demais prezadas “Batatas”: Beatriz, Estevão, Gabriela e Nathan, pelas aconchegantes memórias, pela paciência com as ausências e pelo carinho invariável.

Às famílias de Minas Gerais e de São Paulo, pela sempre amorosa e calorosa torcida.

Aos meus irmãos André e Caio, pela amizade perene e inabalável. Por compartilharem comigo seu crescimento e serem parte fundamental do meu.

Aos inestimáveis pais Marisa e Sandro, por cultivarem despodados meus caminhos, me permitindo errar, desviar, retornar, e descobrir quem eu sou a cada passo. Pelo amor incondicional com que me recebem após cada dificuldade.

À querida Alê, por trazer a aurora mesmo às piores tempestades. τ.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

ABSTRACT

SILVA, P. C. B. I. **On bipartite decision forests**. 2024. 123p. Dissertation (Master in Science) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2024.

The present study investigates decision forest algorithms for predicting interactions in bipartite networks. We concentrate on examples of such problems in the biological domain, such as drug-protein interactions, microRNA-gene interactions or long non-coding RNA-protein interactions. Notwithstanding, the proposed methods encompass the broad range of tasks satisfying i) the goal is to predict interactions between two entities; ii) the interacting pairs are composed of two different types of entities; and iii) each type of entity has its own set of input features. We refer to this paradigm as bipartite interaction learning or *bipartite learning*. Predicting interactions in such networks has fundamental challenges. For instance, the number of possible interactions is often very large in comparison to the number of known interactions. As a result, the data is frequently sparse, and negative annotations are unreliable. We explore a class of decision forest models specifically designed to address these challenges, that we broadly call *bipartite forests*. First, we demonstrate how these trees can be adapted to yield a $\log n$ speedup in training time. We also propose using weighted-neighbors approaches to determine each leaf's output, which resulted in improved generalization. Finally, we introduce semi-supervised impurity functions to bipartite forests. These functions result in trees that also consider clusters of instances in the feature space, rather than only their labels. This is shown to improve the forests' resilience to the missing annotations. Our models display highly-competitive performance across ten interaction prediction datasets. We believe the proposed methods can be a crucial step in developing effective and scalable machine learning models for interaction prediction. Further adaptations of these models could also impact other domains, such as recommendation systems, multilabel learning and weak-label learning.

Keywords: Decision forests. Interaction prediction. Bipartite learning. Positive-unlabeled learning.

RESUMO

SILVA, P. C. B. I. **Florestas de decisão bipartidas**. 2024. 123p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2024.

O presente estudo investiga algoritmos de floresta de decisão para prever interações em redes bipartidas. Concentra-se em exemplos de tais problemas no domínio biológico, como interações fármaco-proteína, interações microRNA-gene ou interações entre moléculas de RNA longo não codificante e proteínas. No entanto, os métodos propostos abrangem uma ampla gama de tarefas de aprendizado, caracterizadas por i) o objetivo é prever interações entre duas entidades; ii) os pares de interação são compostos por dois tipos diferentes de entidades; e iii) cada tipo de entidade possui seu próprio conjunto de características de entrada. Refere-se a este paradigma como aprendizado de interações bipartidas, ou *aprendizado bipartido*. Prever interações em tais redes nos apresenta desafios fundamentais. Por exemplo, o número de interações possíveis é frequentemente muito superior ao número de interações conhecidas. Como resultado, os dados são muitas vezes esparsos, e as anotações negativas são incertas. Exploramos uma classe de florestas de decisão especificamente projetadas para enfrentar esses desafios, que chamamos de *florestas bipartidas* em geral. Primeiro, demonstramos como essas árvores podem ser adaptadas para obter uma melhora logarítmica no tempo de treinamento. Também propomos o uso de abordagens de vizinhos ponderados para determinar a saída de cada folha, resultando em melhora na capacidade de generalização dos modelos. Finalmente, introduzimos funções de impureza semi-supervisionadas para florestas bipartidas. Essas funções resultam em árvores cientes da densidade do espaço de características, em vez de apenas considerar os rótulos para o crescimento. Mostra-se que isso melhora a resiliência das florestas às anotações faltantes. Nossos modelos exibem desempenho altamente competitivo em dez conjuntos de dados de previsão de interação. Acreditamos que os métodos propostos podem ser um passo crucial no desenvolvimento de modelos de aprendizado de máquina eficazes e escaláveis para prever interações. Adaptações adicionais desses modelos também podem impactar domínios vizinhos, como sistemas de recomendação, aprendizado multi-rótulo e aprendizado de rótulos fracos.

Palavras-chave: Florestas de decisão. Predição de interações. Aprendizado bipartido. Aprendizado positivo-não rotulado.

LIST OF FIGURES

Figure 1 – Representation of a bipartite network and its corresponding dataset.
Source: By the author. 33

Figure 2 – The standard global single-output (SGSO) approach. The bipartite dataset is represented as two matrices X_1 and X_2, and a matrix Y of shape $|X_1|_i$ by $|X_2|_j$. The SGSO approach concatenates the rows of X_1 and X_2 to form a new feature matrix, in which each row represents a dyad. The interaction matrix Y is then reshaped into a column vector, so that each element of Y is associated with a row of the new feature matrix. The resulting dataset is then used to train a conventional single-output estimator.
Source: By the author. 35

Figure 3 – The standard global single-output (SLMO) approach. The bipartite dataset is represented as two matrices X_1 and X_2, and a matrix Y of shape $|X_1|_i$ by $|X_2|_j$. The SLMO approach trains two primary multi-output estimators, one for each axis, and then trains two secondary multi-output estimators on the predictions of the primary models. The secondary models are then used to predict the interactions between new row samples and new column samples.
Source: By the author. 36

Figure 4 – A decision tree model. A new sample $\mathbf{x}$ traverses the tree structure from the root node to a leaf, where a final decision is made. Each bifurcation represents a decision rule in the format $\mathbf{x}^{[f]} > t$. The validity of the rule determines if the sample should follow the left or right path. When a terminal node is reached, the output value is generated by the `prototype` function.
Source: By the author. 38

Figure 5 – Different split configurations for validating models with bipartite datasets.
Source: By the author. 58

Figure 6 – Bipartite cross-validation. We represent the 9 train-test splits of a 3 by 3 bipartite cross-validation procedure.
Source: By the author. 59

Figure 7 – Diagonal bipartite cross-validation. We represent the 3 train-test splits of a 3 by 3 diagonal bipartite cross-validation procedure.
Source: By the author. 60

Figure 8 – Point biserial correlation between the binary labels at each percentile rank and the AUROC and AUPR scores. The correlation was calculated between the ranks and the scores for $N = 10^5$ random sets of $R = 10^3$ binary labels. The results show that AUPR is very sensitive to a small group of highest ranks, while AUROC has a more distributed weigthing profile. For AUROC, both higher and lower ranks have a higher impact than ranks around 0.5, contributing in opposite directions to the final metric. The results confirm the theoretical analyses from Section 2.7.3.2.
Source: By the author. 74

Figure 9 – Empirical time complexity estimation of the proposed bipartite global single-output (BGSO) and the global multi-output (GMO) (1) algorithms. Bipartite versions of both extremely randomized trees (2) (BXT) and greedy decision trees (3) (BDT) were built under the BGSO and GMO scheme and trained over artificial datasets of varying numbers of samples (as described in Section 2.4.2). The training time versus the number of samples is presented on the left. On the right, we present the same plot on a logarithmic scale. The slopes and respective standard deviations are obtained by applying least-squares linear regression. Independent two-sample t-tests comparing the slope estimates reveal that the time complexity of `bdt-bgso` is highly significantly lower than `bdt-gmo` ($p < 10^{-64}$) and even `bxt-gmo` ($p < 10^{-20}$), and also that `bxt-gso` significantly exhibits lower complexity than `bxt-gmo` ($p < 10^{-22}$).
Source: By the author. 76

Figure 10 – Comparison of bipartite random forests for different prototype strategies. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 90

Figure 11 – Comparison of bipartite Extra-Trees for different prototype strategies. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure ($4, 5$) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 91

Figure 12 – Comparison of random forests under different adaptation strategies to bipartite interaction data. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure ($4, 5$) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 92

Figure 13 – Comparison of Extra-Trees under different adaptation strategies to bipartite interaction data. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure ($4, 5$) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 93

Figure 14 – Comparison of bipartite random forests with and without label imputation by NRLMF. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 94

Figure 15 – Comparison of bipartite Extra-Trees with and without label imputation by neighborhood-regularized logistic matrix factorization. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 95

Figure 16 – Comparison of semi-supervised BGSO BXT with different strategies for determining the unsupervised impurity and the supervision balance. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 96

Figure 17 – Comparison of different bipartite forests. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4,5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 97
Figure 18 – Comparison of two proposed bipartite forests (`bxt-sq-yr` and `bxt-bgso-yr`) with several prominent models from the literature. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure $(4, 5)$ in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author. 98

CONTENTS

1 INTRODUCTION . . . 21
1.1 What is *bipartite learning*? . . . 21
1.2 Challenges of bipartite learning problems . . . 23
1.3 How can bipartite models be built? . . . 24
1.4 Why decision trees? . . . 26
1.5 Related work . . . 27
1.6 Research questions . . . 28

2 DEVELOPMENT . . . 31
2.1 Definitions . . . 31
2.1.1 Mathematical notation . . . 31
2.1.2 Problem statement . . . 32
2.2 Data-centric adaptations of learning algorithms . . . 33
2.2.1 The standard global single-output adaptation . . . 34
2.2.2 The standard local multi-output adaptation . . . 35
2.3 Decision trees . . . 36
2.3.1 Searching for the best split . . . 39
2.3.2 Measuring the quality of a split . . . 42
2.4 Bipartite decision trees . . . 43
2.4.1 Bipartite global single-output trees . . . 43
2.4.2 Asymptotic complexity analysis . . . 45
2.4.3 Prototype functions for bipartite trees . . . 47
2.5 Decision forests . . . 48
2.6 Incorporating semi-supervision into decision trees . . . 51
2.6.1 Semi-supervised learning . . . 52
2.6.2 Semi-supervised decision trees . . . 53
2.6.3 Unsupervised impurities . . . 54
2.6.4 Heuristics for balancing the supervised and unsupervised objectives . . . 55
2.7 Assessing the performance of bipartite models . . . 56
2.7.1 There are multiple ways to measure model generality . . . 57
2.7.2 Cross-validating in two dimensions . . . 59
2.7.3 Prediction scoring metrics . . . 60
2.7.3.1 Ideal descriptions of AUROC and AUPRC . . . 64
2.7.3.2 AUPR and AUROC in terms of ranked decision values . . . 65
2.7.3.3 AUROC is the normalized mean percentile ranks . . . 68
2.8 Experimental settings . . . 69

2.8.1 Datasets 69
2.8.2 Evaluation procedure 71
2.9 Experiments 72
2.9.1 What are the differences between AUROC and AUPR? 72
2.9.1.1 Setup 73
2.9.1.2 Discussion 73
2.9.2 Are BGSO models faster than GMO models? 74
2.9.2.1 Setup 75
2.9.2.2 Discussion 75
2.9.3 Which prototype should a GMO forest use? 75
2.9.3.1 Setup 77
2.9.3.2 Discussion 77
2.9.4 Which adaptation strategy is the best for decision forests? 78
2.9.4.1 Setup 79
2.9.4.2 Discussion 79
2.9.5 Can label imputation assist bipartite forests? 80
2.9.5.1 Setup 81
2.9.5.2 Discussion 82
2.9.6 What is the best way of building semi-supervised forests? 83
2.9.6.1 Setup 83
2.9.6.2 Discussion 84
2.9.7 Which strategies are the most promising? 85
2.9.7.1 Setup 85
2.9.7.2 Discussion 85
2.9.8 Can bipartite forests compete with other proposals? 86
2.9.8.1 Setup 87
2.9.8.2 Discussion 88

3 CONCLUSION 99
3.1 Main findings 99
3.2 Future work 100

REFERENCES 103

APPENDIX 111

APPENDIX A – OTHER ESTIMATOR-CENTRIC STRATEGIES . . 113
A.1 Linear models 113
A.2 Neighborhood-Regularized Matrix Factorization 115

A.2.1 Traditional matrix factorization 116
A.2.2 Neighborhood regularization 118

1 INTRODUCTION

As the starting point, we provide a brief overview of the concepts explored along this thesis. We begin by describing the learning paradigm of interest (Section 1.1), and proceed by outlining its main challenges and current approaches (Section 1.2 and Section 1.3). A brief discussion on the advantages of decision trees is then presented (Section 1.4), followed by a summary of previous literature on bipartite forests (Section 1.5). We conclude this introduction by presenting the guiding research questions of our work (Section 1.6).

1.1 What is *bipartite learning*?

The effect a drug molecule has on our organism is tightly associated with the specific microscopic structures it physically and chemically interacts with (6, 7). The regulatory roles of microRNAs are often mediated by their binding to a defined set of messenger RNAs. The activity of a transcription factor is determined by which genes it is capable of regulating (8). The success of a recommendation system depends on its ability to predict a user's preference for a set of items in a catalog (9). All those scenarios share the same underlying structure: there are two distinct domains of objects that interact with one another, forming a heterogeneous *bipartite network* (10).

Frequently, great importance lies in computationally describing such relationships. For instance, developing new drug molecules is a costly and time-consuming process, requiring years of complex computer simulations, large screening assays, and extensive trials (6, 7). Being able to predict beforehand the interactions in a large dataset of drugs and protein targets holds a remarkable potential to accelerate drug development and reduce its costs, enabling the concentration of financial resources and human expertise to a set of most promising candidates. Furthermore, being able to characterize on a large scale the relationships between microRNAs and genes or other biomolecular entities can greatly assist the understanding of regulatory mechanisms of gene expression, a valuable step towards the discovery and development of new therapies.

Therefore, it is natural that numerous machine learning algorithms have been proposed to predict interactions in bipartite networks (6, 7, 11). The role of a learning algorithm in this setting is to receive a feature vector describing the heterogeneous pair, and characterize their interaction through a binary label or affinity score. We focus on cases where the input vector can be split into two: one vector representing each interacting entity. More specifically, our setting is characterized by:

1. **Interaction prediction:** The goal is to infer interactions that occur each between two objects (called a *dyad*).

2. **Heterogeneus dyads:** Each dyad is composed of objects of different types.

3. **Bipartite network:** There are two types of objects.

4. **Available side-features:** Each object has its own feature vector that is not derived from its set of known outputs.

Several similar concepts have emerged when referring to learning tasks that aim to describe relationships. *Link prediction* (12,13) refers to the task of predicting the existence of connections between two nodes in a network. Usually, is assumes that node attributes are unavailable, being concerned with recovering attributes from the network structure (12). Furthermore, the output is usually binary (or probabilistic), indicating the presence or absence of a link (12). *Dyadic prediction* (14–16) is a broader category of tasks involving predicting relationships between two objects. The availability of side-features is considered, but not required (14), and possible outputs encompass both classification and regression (14). The presence of two separate domains is sometimes assumed (14, 15), but there are also cases in which the network is homogeneous (16). *Interaction prediction* is a term often used in the context of biological networks (6,7,17,18). As such, it usually assumes the presence of side-features. The output can be either a binary indicator or real values such as drug-target affinities (19).

None of these terms are specific to bipartite networks. They include cases in which a single type of entity composes the network, and thus a single feature matrix is sufficient to describe all the nodes. Furthermore, it is common for these contexts to assume *transductive* (20) approaches. In such cases, the attributes of all nodes are available for training, and only the set of links is split between training and validation sets (12, 13). As a result, the models are not guaranteed to be able to predict interactions between unseen instances.

In this study, we are mostly concerned with the bipartite scenario, where two distinct sets of entities interact with each other and each set has its own feature matrix. We are also interested in *inductive* (20) techniques, in which the main goal is to model a function from the node feature space to the output space. This property allows us to predict interactions between new instances that were not present in the training set, while also enabling transparent models to provide new insights about the underlying phenomena.

If feature vectors are not available for both interacting entities, the problem becomes essentially a multi-output supervised task or a transductive dyadic prediction problem. For that reason, we require side-features to be available for both axes.

Possible denominations for the current setting include *inductive heterogeneous dyadic prediction* and *inductive bipartite interaction learning*. For simplicity, we refer to this paradigm as *bipartite learning*.

1.2 Challenges of bipartite learning problems

Despite their importance, bipartite interaction prediction problems pose unique challenges to machine learning approaches, that we enumerate below.

1. **Heterogeneous data:** The input data is often composed of two very dissimilar types of objects, each with its own set of features. As a result, estimators may be required to simultaneously deal with descriptors of very different natures (6).

2. **Large datasets:** Due to the intrinsic combinatory nature of this type of problem, the interactions to be processed are usually very numerous, even if the number of training instances in each domain is not that expressive. Specifically, for each new object introduced to the training set, a new interaction could be considered for each training sample in the other domain, rendering the number of interactions to grow quadratically with the number of entities of each type.

3. **Knowledge sparsity:** As a consequence of the previous point, no amount of research efforts in characterizing new interactions can keep up with the rate at which possible relationships appear, resulting in a fundamental sparsity of confidently verified interactions (6).

4. **Lack of validated negative annotations:** In many cases, the absence of a relationship between two objects is much harder to validate than its presence. For instance, negative results in biochemical essays asserting molecular interactions can often be caused by a plethora of external experimental factors other than the actual lack of interaction under optimal conditions, unlike positive results that usually imply a successful assay (6, 7, 19). These factors result in a general lack of high-quality negative data in interaction datasets, requiring special considerations from machine learning pipelines that are often overlooked in the literature.

5. **Impact on model evaluation:** Two rank-based metrics are mainly used to evaluate the performance of interaction prediction models (7, 12, 13, 19): the area under the receiver operating characteristic curve (AUROC) (21, 22) and the area under the precision-recall curve (AUPR) (21,23,24). These metrics are known to assess the performance of a model in different ways, and are often used in conjunction to provide a more comprehensive view. However, these differences are not fully clear, and neither is how they are affected by the missing annotations. This makes it difficult to interpret why different models are sometimes chosen, and to decide which metric to prioritize in each experiment. For instance, AUPR is often recommended for imbalanced scenarios (11,21,22,25), but metrics similar to AUROC are also suggested in interaction prediction contexts (11, 19, 26–28).

6. **Lack of inductive algorithms:** Many of the algorithms proposed for interaction prediction assume a *transductive* (20) setting, where attributes of the test instances are available

at training time (11, 19). This setting hinders the interpretability of those models and their application to completely new entities.

Most of these issues are an integral part of interaction prediction problems, that arise from how they are fundamentally defined around combinations of objects. However, these differences are often overlooked when developing machine learning algorithms (19). For instance, it is common to represent each dyad with the concatenated feature vectors of its components. Nevertheless, it can be challenging to collect representations for all dyads in this format, which would result in a very large matrix to be processed by the algorithm. Instead, the training set for a bipartite problem is more naturally represented as two separate design matrices rather than the usual single one. Ideally, we would want to avoid the combination of feature vectors as a data preprocessing step, leaving the conjoint treatment of both design matrices as a matter of estimator design. This view defines a different machine learning paradigm, in which what we call "instances" or "samples" are not the dyads themselves, but rather each individual interacting object. The estimator receives both feature matrices and it is up to the algorithm to decide how to use them in conjunction to predict new interactions.

In conclusion, acknowledging such defining aspects and incorporating the bipartite nature of the problem are crucial steps in developing successful and scalable machine learning models for interaction prediction.

1.3 How can bipartite models be built?

There are two general strategies for developing machine learning algorithms for bipartite interaction prediction. We call them *data-centric* and *estimator-centric* approaches. The majority of methods proposed for bipartite learning is based on organizing the data to enable the use of traditional learning algorithms. These data-centric adaptations can be broadly classified into two categories (1, 17, 29), which we describe below.

- **Global single output:** The most common approach observed in the literature. The dataset is formatted so that each dyad is an instance. Each dyad has a single label associated with it, and the feature vector describing the dyad is the concatenation of the two feature vectors describing the interacting entities. Usual single-output learning algorithms can then be applied to the problem.
  - **Main advantage:** Enables the estimator to consider features from both instance domains simultaneously.
  - **Main disadvantage:** It can be computationally infeasible to build the feature vectors and train the estimator for all possible interactions.
- **Local multi-output:** This strategy separates the bipartite learning problem into two multi-output tasks. We first select one of the domains as input and the other as output. We train

a multi-output estimator to receive the features of the input domain and predict its interactions with the output domain. Each instance in the output domain is considered an output to me modeled, so that the features of the output domain are completely disregarded in this step. The same procedure is then repeated with the roles of the domains reversed. Predicted interactions of these two models are fed into a new round of training that occurs in a similar fashion. A total of four models is necessary to predict interactions between instances that were both not present in the training set.

  - **Main advantage:** A much smaller number of input samples is considered, in comparison to the global single output approach. As a consequence, the models are usually faster to train.
  - **Main disadvantage:** It requires training new models for each new batch of interactions to be predicted. This severely limits the application on online learning settings.

Other methods adapt the learning algorithm itself to take full advantage of the bipartite format. These strategies tend to be more computationally efficient than the global single output approach, since they consider the dataset in a more compact format. Notwithstanding, they are still able to consider both domains simultaneously, unlike the local multi-output procedure. There is, however, a limited number of learning algorithms that can be adapted in a more fundamental level. Notable examples of estimator-centered adaptations are listed below.

- **Linear regression:** Instance-instance similarities among each domain could be used to build a dyad-dyad similarity matrix. This matrix could then be used as a kernel matrix for training a linear regression model in a global single output fashion. The main problem with this idea is that a dyad-dyad kernel matrix would be often prohibitively large. Kron-RLS (30) solves this issue, building the hypothetical linear model we described without explicitly obtaining the dyad-dyad kernel matrix.
  - **Main advantage:** It is a remarkably efficient method.
  - **Main disadvantage:** Linear relationships frequently are too simplistic to capture the complexity of the underlying interactions.
- **Matrix factorization:** The adjacency matrix of the bipartite network is decomposed into two latent feature matrices, one for each domain. The product of these matrices approximates the original adjacency matrix, and can be used to predict new interactions between known instances. 31 introduces a method to extend this idea to predict interactions with new instances, by encouraging instance-instance vicinities to be transfered from the original to the latent feature space.
  - **Main advantage:** It is a naturally scalable technique, being designed from the ground up for interaction prediction tasks.

  - **Main disadvantage:** Interpretability is limited and a large number of hyperparameters is necessary.

- **Decision trees:** 1 proposes an adaptation to the split search procedure occurring at each decision tree node. The search is performed locally for each domain, similarly to the local multi-output approach, but the resulting model is a single decision tree.
  - **Main advantage:** Better scores are achieved in comparison to decision trees adapted with data-centric approaches (1).
  - **Main disadvantage:** No improvement in training complexity is observed relative to decision trees under the global single output adaptation (1).

This work will focus on decision trees. We explore how tree-based algorithms can be adapted to the bipartite learning paradigm, and how they can be tailored to address the challenges imposed by our learning setup.

1.4 Why decision trees?

Decision trees are a very interesting option to explore. They show remarkable interpretability, flexibility to different tasks, and straightforward usage, besides demonstrating remarkable predictive power when combined into ensembles (32–35). We list below some of the main advantages of learning algorithms based on decision trees.

1. **Interpretability:** Decision trees and forests are well-known transparent models, with several methods being available to interpret their predictions (3,36,37). For instance, the frequency with which a feature is used by the tree can reflect its importance in the prediction process (3).
2. **No need for preprocessing:** In most cases, very little preprocessing is necessary to use decision trees. They can directly handle continuous or categorical features, and natively deal with missing feature values (3). They are also not sensible to feature scaling, so normalization is most often not necessary.
3. **Flexibility:** Decision trees can be used for both classification and regression tasks, and can handle either single or multi-output problems. They were successfully adapted to a wide range of scenarios (35).
4. **Small number of hyperparameters:** They have a small number of hyperparameters in comparison to other algorithms such as deep neural networks, which makes them easy to use and tune (34).
5. **Predictive power:** They can uncover complex non-linear relationships between features and labels, while being notably resistant to overfitting when combined into ensembles.

They are widely recognized for their predictive performance especially on structured data (35).

6. **Fast inference:** Once trained, they can make predictions in logarithmic time complexity (3).

We thus investigate how the potential of decision forests can be applied to bipartite learning. We introduce three main methodology refinements, aimed at the high dimensionality, sparsity, and lack of information imposed by our learning settings.

1. **More scalable trees:** We experiment with different metrics to guide the tree growing procedure. Optimizing the algorithm for these metrics can offer improvements in training complexity, making bipartite trees more scalable.
2. **Weighted neighbors prototypes:** We explore using similarity scores between instances to calculate the output value of each leaf node. This enables shallower trees to be built while increasing their ability to generalize.
3. **Semi-supervised impurities:** We build trees that consider how the instances are grouped in the feature space, and not only their labels. This improves the model's resilience to missing annotations.

1.5 Related work

This section provides a brief overview of previous investigations related to bipartite forests, pointing out limitations to be addressed in this work.

17 stablishes the idea of the global single output, local single output, and local multi-output adaptations of traditional machine learning algorithms. In their study, they analyse the performance of random forests (RF) (32) and ensembles of randomized trees (ERT) (2) adapted in both ways and applied to biological interaction prediction. They consider not only bipartite but also homogeneous networks, such as protein-protein interactions or gene-gene interactions (17).

In 2018, Pliakos, Geurts and Vens(1) present a global multi-output decision tree for interaction prediction, named the predictive bi-clustering tree (PBCT) algorithm. The method adapts decision trees on a more fundamental level to interaction scenarios, resulting in improved results over the previous adaptations (1). However, the training complexity of the proposed trees is still the same as that of the previous adaptations (1), and the authors do not explore the use of semi-supervised techniques.

The authors, in 2019, extend the PBCT algorithm to ensembles, building bipartite versions of RF and ERT (18). They demonstrate superior performance against the other models in the comparison. Later, in 2020, they explore the use of neighborhood-regularized logistic matrix factorization (NRLMF) (31) as a pre-training step to the bipartite ERT (29). NRLMF is used to generate continuous pseudo-labels for negative interactions, that are imputed in the interaction matrix before training the forest estimator. They demonstrate significant improvements in prediction scores resulting from this procedure (29), sugesting that label imputation and other semi-supervised techniques could be beneficial.

38 extends the application of PBCTs to hierarchical multi-label classification problems. In ths context, the bipartite trees are able to split apart not only samples but also labels, resulting in competitive predictive performance.

39 and 40 independently propose a way to consider semi-supervised assumptions when building decision trees. They do so by penalizing the selection of split points that separate close training instances. As a result, instances within the same tree node, that tend to yield the same output, also will tend to have similar feature vectors. The method is based on the assumption that neighboring instances are more likely to have similar labels (the *smoothness* assumption (20, 41)). The impact of missing or incorrect labels on the split selection is then reduced, since we delegate part of their influence to the feature space directly. While 39 uses variances of features as the impurity function, 40 uses the structure of random trees to determine similarity values (42). Neither of these works, however, is centered in interaction prediction.

In 2023, 43 employed semi-supervised impurities with bipartite trees. To enhance performance, the authors do not use the semi-supervised impurity to evaluate every possible split. Instead, they first determine the best split of each of each feature domain in a supervised manner, and then use the semi-supervised impurity to choose between the two best supervised splits. This procedure is performed at each tree node. The algorithm results in better predictions than the fully supervised approach, while being faster than the original semi-supervised proposal (43). The authors also propose a different way to balance the influences of the supervised and unsupervised objectives. They dynamically determine new weights for each objective in each tree node, based on the amount of labeled data available (43). Nonetheless, the authors do not explore ensembles of bipartite trees, or the more expensive semi-supervised evaluation of every candidate split. They also do not explore other heuristics for dynamic supervision.

1.6 Research questions

The present work will explore current open questions regarding bipartite forests. We explore how these algorithms can be improved both in terms of training complexity and predictive performance, and how they compare with other prominent techniques for interaction prediction. We enumerate our main research questions below.

1. **Can bipartite trees be faster?**

   Investigate how can we exploit the bipartite nature of the problem to achieve a speedup in the training of decision trees.

2. **Are semi-supervised techniques beneficial for bipartite forests?**

   Test the hypothesis that semi-supervised techniques can improve the resilience of bipartite trees to missing annotations.

3. **How do AUROC and AUPR differ in their assessment of model performance?**

   Evaluate in detail the differences between the two most common threshold-agnostic metrics for binary classification tasks: the Area Under the Receiver Operating Characteristic Curve (AUROC) and the Area Under the Precision-Recall Curve (AUPR).

4. **How do bipartite forests compare with proficient models in the field?**

   Compare the performance of bipartite forests with other state-of-the-art models for bipartite learning.

2 DEVELOPMENT

This chapter presents a detailed description of the proposed methods and experimental results of this work. To facilitate future reference, we dispose all the experiments in a single section. However, the reader is encouraged to proceed in a more modular fashion when reading this thesis in its entirety. We provide boxes at the end of several sections listing the experiments that relate to what was discussed. The reader might opt for skipping to the relevant experiments before proceding. If necessary, more details on the evaluation procedure can be found in Section 2.7 and Section 2.8.

The sections in this chapter are organized as follows. We begin by formally defining the notation we utilize and the learning problem at hand (Section 2.1). Section 2.2 will then present the most common strategies for building machine learning models for bipartite interaction prediction. Further, we provide a general description of decision tree algorithms (Section 2.3), as the theoretical foundation for the subsequent formal definition of bipartite decision trees (Section 2.4). Section 2.5 will then present how decision trees can be combined into powerful ensembles. Section 2.6 discusses strategies for acknowledging the label uncertainty of our problems, leveraging assumptions from semi-supervised learning to fundamentally influence how decision trees are grown. We then present and discuss the model evaluation process in the bipartite setting (Section 2.7), pointing out its specificities and discussing implications for two common scoring metrics. Finally, our experimental results are gathered by Section 2.9.

2.1 Definitions

This section defines the notation used throughout this work (Section 2.1.1) and formalizes the learning tasks we approach (Section 2.1.2), defining how our datasets will be represented.

2.1.1 Mathematical notation

We use lowercase letters for scalars, bold lowercase letters for vectors and uppercase letters for bidimensional matrices. For any given matrix M, we denote by $M^{[ij]}$ its element on the i-th row and j-th column ($i, j \in \mathbb{N}^*$). The vector representing the i-th row of M is denoted $M^{[i]}$. We represent by $M^{[\cdot j]}$ the vector $(M^\intercal)^{[j]}$ referring to the j-th column of M. Defining the index notations as superscripts enables the subscripts to be used only as indentifiers, naming the matrix or vector as a whole and not each element. Indices are also always represented by a single letter, to avoid the need for separators between them.

Vector concatenation or horizontal concatenation of matrices is expressed as $\begin{bmatrix} A & B \end{bmatrix}$.

Vertical concatenation is expressed as $\begin{bmatrix} A \\ B \end{bmatrix}$.

We write the total number of rows in M as $|M|_i$ and its number of columns as $|M|_j$. The total number of elements in M is written $|M| = |M|_j|M|_i$, not to be confused with the determinant of M.

We display filtered matrices or vectors by writing the condition as the index, optionally enclosed by parentheses when necessary or improving readability:

$$M^{[(i<3)j]} = \{M^{[kj]} \mid k < 3\}$$

When summing over all indices in a given dimension, we take the liberty of omitting the start and end positions:

$$\sum_i M^{[ij]} = \sum_{i=1}^{|M|_i} M^{[ij]}$$

We also represent averages in a more concise way:

$$\begin{aligned} M^{\langle i\rangle [j]} &= \frac{\sum_i M^{[ij]}}{|M|_i} \\ M^{[i]\langle j\rangle} &= \frac{\sum_j M^{[ij]}}{|M|_j} \\ M^{\langle ij\rangle} &= \frac{\sum_j \sum_i M^{[ij]}}{|M|} \end{aligned} \tag{2.1}$$

Scalar exponents represent element-wise exponentiation of matrices or vectors. We also omit parentheses in some situations, as exemplified by Equation 2.2.

$$\begin{aligned} M^{[ij]2} &= (M^{[ij]})^2 \\ M^{\langle ij\rangle 2} &= (M^{\langle ij\rangle})^2 \\ M^{2\langle ij\rangle} &= (M^2)^{\langle ij\rangle} \\ M^{\langle i\rangle 2\langle j\rangle} &= ((M^{\langle i\rangle})^2)^{\langle j\rangle} \end{aligned} \tag{2.2}$$

2.1.2 Problem statement

Consider two object spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ of different natures, and an output space $\mathcal{Y}$. Bipartite interaction prediction is the task of modeling a function $f \colon \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{Y}$.

Naturally, f is not directly available. To build the model, we only have access to a *dataset* $\mathcal{D}$ containing n_1 instances $\mathrm{x}_1 \in \mathcal{X}_1$, n_2 instances $\mathrm{x}_2 \in \mathcal{X}_2$, and all their corresponding outputs $\mathbf{y} = f(\mathrm{x}_1, \mathrm{x}_2)$. That is, $\mathcal{D} = \{(\mathrm{x}_{1,i},\ \mathrm{x}_{2,j},\ \mathbf{y}_{i,j}) \mid 1 \le i \le n_1,\ 1 \le j \le n_2\}$. Unfortunately, the output $\mathbf{y}$ is seldom fully available, as discussed in Section 1.1, but we keep the same formulation by letting $\mathbf{y}$ assume identifiers for missing entries. The available objects represented by the vectors x_1 and x_2 are called *samples* or *instances*, so that $\mathcal{X}_1$ and $\mathcal{X}_2$ are also

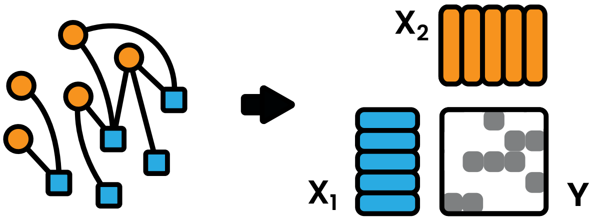

Figure 1 – Representation of a bipartite network and its corresponding dataset.
Source: By the author.

called sample or instance *domains*. The vector representations themselves, $\mathbf{x}_1$ and $\mathbf{x}_2$, are called *feature vectors* or *attribute vectors*. Any heterogeneous pair of instances $(\mathbf{x}_1,\ \mathbf{x}_2)$ is called a *dyad*. Each element of the output $\mathbf{y} \in \mathcal{Y}$ is called a *label* or a *target*. The procedure that takes $\mathcal{D}$ and generates the model for f is called the *learner*, the *learning algorithm* or the *estimator*.

In our specific case, $\mathcal{X}_1 = \mathbb{R}^{m_1}$, $\mathcal{X}_2 = \mathbb{R}^{m_2}$, and $\mathcal{Y} = \{0, 1\}$, so that f is a binary single-output classifier. $\mathcal{D}$ can then be represented by a set of matrices: $\mathcal{D} = (X_1,\ X_2,\ Y)$, where $X_1^{[ij]} = \mathbf{x}_{1,i}^{[j]}$, $X_2^{[ij]} = \mathbf{x}_{2,i}^{[j]}$, and $Y^{[ij]} = y_{i,j}$ (see Figure 1). We call X_1 and X_2 *feature matrices*, *attribute matrices* or *design matrices*. Y is called the *label matrix*, the *interaction matrix*, or the *adjacency matrix*. Since each sample in X_1 refers to a *row* of Y and each sample in X_2 corresponds to a *column* of Y, we sometimes refer to the samples of X_1 and X_2 as the *row samples* and the *column samples*, respectively. Analogously, we say that our datasets have m_1 *row features* and m_2 *column features*.

Another important characteristic of our learning problems is that only the positive labels ($y = 1$) are reliable. This means that an interaction between two instances may occur in reality even if annotated as negative ($y = 0$) in the dataset. The datasets are thus termed positive-unlabeled (PU) (44).

2.2 Data-centric adaptations of learning algorithms

As discussed in Section 1.3, bipartite interaction problems fundamentally differ from the usual machine learning paradigm, in which input data represents a single entity to be labeled. Interaction tasks are instead concerned with labeling a relationship between two entities, and as such, each prediction is made upon a pair of feature vectors, each vector being specific to each of the two sample domains.

Such subtlety is often bypassed by reformulating a bipartite dataset into the traditional machine learning setting (17, 19, 45). These strategies can be encompassed under two general approaches, termed *global* and *local* (1, 12, 15, 29, 45). For the sake of clarity and generality, we further specify these categories by defining *global* and *local* as general properties of estimators, rather than specific training procedures:

- ***Global*** estimators are those aware of both instance domains during the training procedure

(X_1 and X_2).

- ***Local*** estimators are those which only have access to feature information from one of the two instance domains during training (either X_1 or X_2). As such, they are often employed in compositions, combining the predictions of several local models to produce the final output.

Furthermore, to be consistent with (1, 17, 18, 29), we assume the following definitions in our current context of bipartite interactions:

- ***Single-output*** estimators are those which consider all labels, i.e. all $Y^{[ij]}$ elements, irrespectively of the column j or row i they are in. They are all regarded as a *single type of output*.
- ***Multi-output*** estimators, on the other hand, are those which consider each instance, each row or column of Y, as a separate task, for example by defining a composite loss function formed by the combination of losses over each row or column.

Notice that the label matrix Y can still be represented in two dimensions even if the model is single-output in this sense, contrary to the usual case where bidimensionality of Y is a defining characteristic of a multi-output problem.

Finally, the two most common ways of adapting traditional estimators to bipartite data are then named *global single-output* and *local multi-output*, as proposed by 17. We further denote them *standard*, to clearly distinguish them from new adaptation proposals that could be described as LMO or GSO, but work in an entirely different way. Specific definitions and shortcomings of the SGSO and SLMO procedures are now presented.

2.2.1 The standard global single-output adaptation

Arguably the most straightforward way to provide bipartite data into a conventional learning algorithm is by presenting concatenated pairs of row-column samples, labeled by each element of Y. The interacting dyad is now what we consider a sample, and each feature vector is the combination of the feature vectors of the two interacting instances. This is usually done by converting the two X matrices and the interaction matrix Y to a single design matrix we call X_{SGSO} and a column-vector of labels we refer to as Y_{SGSO}.

One way of doing so is by choosing indices as described by Equation 2.3, where all combinations of i_1 with i_2 are explored (see Figure 2).

$$\begin{aligned} i_g &= (i_1 - 1)|X_2|_i + i_2 - 1 \\ X_{\text{SGSO}}^{[i_g]} &= \begin{bmatrix} X_1^{[i_1]} & X_2^{[i_2]} \end{bmatrix} \\ Y_{\text{SGSO}}^{[i_g 1]} &= Y^{[i_1 i_2]} \end{aligned} \tag{2.3}$$

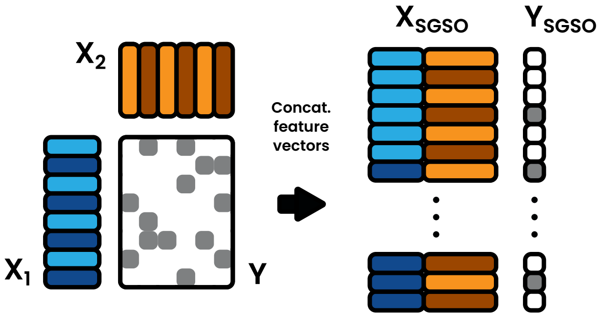

Figure 2 – The standard global single-output (SGSO) approach. The bipartite dataset is represented as two matrices X_1 and X_2, and a matrix Y of shape $|X_1|_i$ by $|X_2|_j$. The SGSO approach concatenates the rows of X_1 and X_2 to form a new feature matrix, in which each row represents a dyad. The interaction matrix Y is then reshaped into a column vector, so that each element of Y is associated with a row of the new feature matrix. The resulting dataset is then used to train a conventional single-output estimator.
Source: By the author.

To consider all possible dyads, X_{SGSO} would have $|X_1|_i|X_2|_i$ rows and $|X_1|_j + |X_2|_j$ columns. Thus, X_{SGSO} as defined by Equation 2.3 is highly redundant. As a result, dealing with such a large X_{SGSO} matrix is impeditive in many cases, both in terms of memory usage and computation time. Therefore, a commonly used workaround is to undersample the negative annotations, yielding a dataset with equal amounts of positive and negative interactions (46, 47). This strategy is justified by the negative-annotated dyads being usually far more numerous than positives and much more likely to be truly negative interactions than false negatives. Nonetheless, we demonstrate in Section 2.9.4 that undersampling negatives is significantly detrimental if the main goal is to indicate wich interactions are most likely to be true.

2.2.2 The standard local multi-output adaptation

The local approaches, in contrast to global methods, propose training different models on X_1 and X_2, so that each estimator only has access to information regarding either row samples or column samples.

As such, multiple non-bipartite models need to be used in conjunction to predict interactions between new row samples and new column samples. The standard local multi-output (SLMO) approach uses two traditional estimators for each axis (four in total), that we here refer to as *primary* and *secondary* estimators. In general, they must be multi-output estimators, each being able to output a bidimensional Y matrix.

The procedure for training the estimators in an SLMO setting consists of simply training the primary estimators. The primary rows estimator is trained on X_1 and Y, and the primary columns estimator on X_2 and $Y^\intercal$ (Function TrainLocalModel). The prediction step is however more complicated, involving the training of the secondary estimators on predictions of the primary models. The procedure is described in details by Function PredictLocalModel and illustrated by Figure 3. The `combine` function used in line line 15 of the Function PredictLocalModel

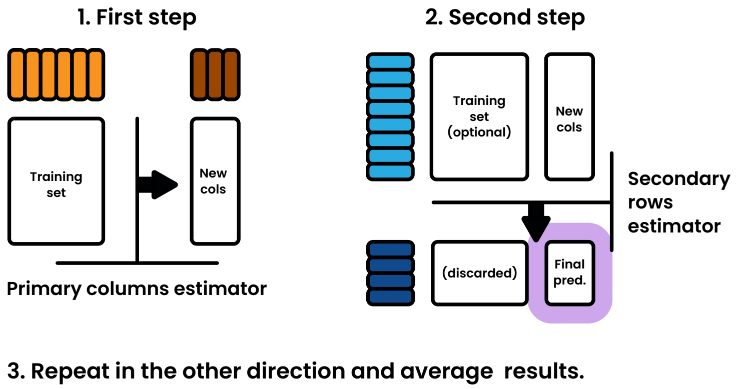

Figure 3 – The standard global single-output (SLMO) approach. The bipartite dataset is represented as two matrices X_1 and X_2, and a matrix Y of shape $|X_1|_i$ by $|X_2|_j$. The SLMO approach trains two primary multi-output estimators, one for each axis, and then trains two secondary multi-output estimators on the predictions of the primary models. The secondary models are then used to predict the interactions between new row samples and new column samples.
Source: By the author.

```
Function TrainLocalModel($X$, $Y$)
    Input: The bipartite training dataset.
    Output: A bipartite local model.
    train(primaryRowsEstimator, $X_1$, $Y$);
    train(primaryColumnsEstimator, $X_2$, $Y^\intercal$);
    return primaryRowsEstimator, primaryColumnsEstimator
```

procedure can be arbitrarily chosen, and is usually defined as the simple element-wise average of both matrices (`combine`$(Y_1, Y_2) = \frac{1}{2}(Y_1 + Y_2)$).

An unusual behaviour occurs if the secondary estimators are able to exploit inter-output dependencies. In this case, the outputs of the SLMO composition would depend on the amount of test data and specific combinations of test instances provided. This is usually not the case in traditional learning problems, and complicates the evaluation procedure. The behaviour forces us to consider the size and sampling method of the batches of test data. For online aplications, the ideal strategy would be to retrain the secondary models for each batch of new instances, using the predictions from the primary models over all the previously received instances (labeled and unlabeled). As a result, the application of SLMO models is effectivelly restricted to offline learning contexts.

2.3 Decision trees

A general description of top-down decision trees is now presented, as a theoretical foundation for the upcoming formal definition of bipartite decision trees (Section 2.4).

Let's consider the scenario where a single protein of interest is selected, and we receive the task of determining which drug molecules will likely affect its physical structure or catalytic

```
Function PredictLocalModel(primary models, $X_{\text{new}}$)
    Input: The trained primary models and the unseen sample matrices $X_{\text{new}}$ for both axes.
    Output: $Y_{\text{pred}}$ predictions for each interaction provided.
    $Y_{\text{new rows}} \leftarrow$ predict (primaryRowsEstimator, $X_{1,\ new}$)
    $Y_{\text{new cols}} \leftarrow$ predict (primaryColumnsEstimator, $X_{2,\ new}$)
    if Secondary estimators consider label dependencies then
        // Concatenate known rows and columns labels to the primary predictions
        $Y_{\text{new cols}} \leftarrow \begin{bmatrix} Y^\intercal \\ Y_{\text{new cols}} \end{bmatrix}$;
        $Y_{\text{new rows}} \leftarrow \begin{bmatrix} Y \\ Y_{\text{new rows}} \end{bmatrix}$;
        // Otherwise, if label columns are considered independently, this step is not necessary
    end
    train(secondaryRowsEstimator, $X_1$, $Y^\intercal_{new\ cols}$);
    train(secondaryColumnsEstimator, $X_2$, $Y^\intercal_{new\ rows}$);
    $Y_{\text{pred rows}} \leftarrow$ predict (secondaryRowsEstimator, $X_{1,\ new}$);
    $Y_{\text{pred cols}} \leftarrow$ predict (secondaryColumnsEstimator, $X_{2,\ new}$);
    if Secondary estimators consider label dependencies then
        // Skip predictions not referring to $X_{1,\ \text{new}}$ and $X_{2,\ \text{new}}$
        $Y_{\text{pred rows}} \leftarrow Y_{\text{pred rows}}^{[\cdot\ (j > n_1)]}$;
        $Y_{\text{pred cols}} \leftarrow Y_{\text{pred cols}}^{[\cdot\ (j > n_2)]}$;
    end
    return combine ($Y_{pred\ rows}$, $Y^\intercal_{pred\ cols}$)
```

function. We wish to find a systematic procedure to decide whether a given drug molecule $\mathbf{x}_i$ will interact with our protein or not. To develop such a procedure, consider we have at our disposal a set of m known drug molecules, whose degree of interaction with our protein of interest was previously experimentally determined. We can then describe a drug molecule $\mathbf{x}_i$ in general by how similar it is to each of our m known molecules, organizing this information as a vector $\mathbf{x}_i = \begin{bmatrix} \mathbf{x}_i^{[1]} & \mathbf{x}_i^{[2]} & \mathbf{x}_i^{[3]} & \cdots & \mathbf{x}_i^{[m]} \end{bmatrix}$ so that $\mathbf{x}_i^{[j]}$ represents the similarity score between the drug $\mathbf{x}_i$ and the j-th of our m known drugs.

The hypothetical decision procedure we intend to determine could then be structured as a path with consecutive bifurcations. We always start at the same place, and, at each bifurcation, a question is asked about the drug $\mathbf{x}_i$ in hands. The questions are in a standard format, exemplified by "Is $\mathbf{x}_i$ more than 60% similar to the 3rd known drug?", or $\mathbf{x}_i^{[j]} > t$, for a general known drug j and similarity threshold t. The answer to the question in each bifurcation deter-

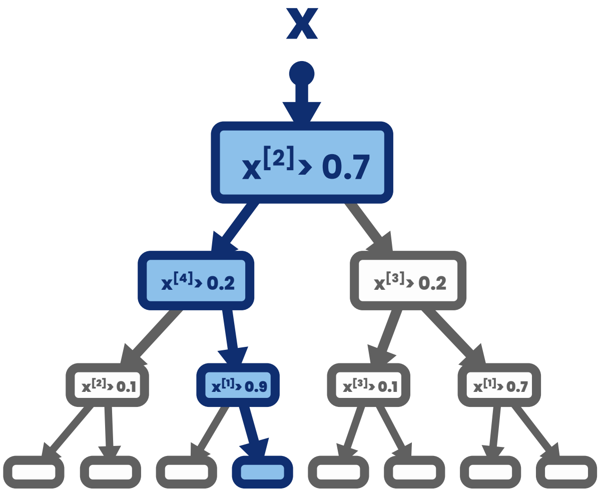

Figure 4 – A decision tree model. A new sample $\mathbf{x}$ traverses the tree structure from the root node to a leaf, where a final decision is made. Each bifurcation represents a decision rule in the format $\mathbf{x}^{[f]} > t$. The validity of the rule determines if the sample should follow the left or right path. When a terminal node is reached, the output value is generated by the `prototype` function.
Source: By the author.

mines which of the two possible paths we should follow. No cycles are allowed in the path, and eventually, all routes reach final locations instead of bifurcations. Each final location contains a fixed value that will be returned as the final decision about the drug $\mathbf{x}_i$'s effect on our protein of interest.

Such a decision procedure, structured as a binary tree path, is what is commonly referred to as a *decision tree* (DT) model (3), illustrated by Figure 4. In this context, each bifurcation then represents a tree *node*, and the final locations are called *leaves*. The value outputted by each leaf is termed the leaf's *prototype*.

The main challenge, however, lies in the building process of such models: in how to determine the rules that define each fork and how we define the stopping criteria for a final decision to be yielded.

To build the decision tree, we are given a training set composed of the two bidimensional matrices X and Y. The dataset represents $|X|_i = |Y|_i$ instances, each described by $|X|_j$ numeric features and labeled by $|Y|_j$ labels. In the previous example, $|X|_i$ would be the number of drug molecules in the dataset $|X|_j$ would be the number of reference drugs (used to obtain the similarities), and $|Y|_j$ would be the number of proteins of interest.

Consider now executing the prediction process of a decision tree for each training instance, going through the branched path. Each bifurcation would divide the training instances between those who answer the question affirmatively and those who answer negatively. Eventually, each leaf will contain a partition of the training instances. From the resulting partitions, we can evaluate the decision tree: a "good" decision tree would be one in which the prototype value of each leaf is a good estimate of the labels in the leaf's partition. The training algorithm uses such evaluation to search for an effective tree. A greedy top-down procedure is usually followed, that we introduce bellow.

Formally, each rule is encoded by an index f representing a feature column and a threshold value t. Each rule then represents a split of the training dataset in two partitions, named *left* and *right*, as defined by Equation 2.4.

$$\begin{aligned} Y_{\text{left}} &= Y^{[(X^{[if]} \leq t)]} \\ Y_{\text{right}} &= Y^{[(X^{[if]} > t)]} \\ X_{\text{left}} &= X^{[(X^{[if]} \leq t)]} \\ X_{\text{right}} &= X^{[(X^{[if]} > t)]} \end{aligned} \tag{2.4}$$

To build a decision tree, we start with the whole training set (the *root node*). A procedure FindSplit is executed to yield a good splitting rule. The training data is then partitioned according to this rule, as defined in Equation 2.4. Two descendant nodes are created, each receiving one of the partitions. For each new node, we have two options: i) apply FindSplit to its data partition and continue recursively; or ii) set the node as a leaf, taking record of the partition it received.

The tree-building algorithms can thus be described by four key components:

1. **The FindSplit procedure**, which determines a set of candidate splitting rules and selects the best one according to a quality metric.
2. **The split quality metric** used by FindSplit to evaluate the candidate rules.
3. **The stopping criteria**, that determine when a node should be set as a leaf.
4. **The prototype function**, that determines the output value of a leaf.

Function BuildTree describes in detail how these components come together to build a decision tree. Function Predict then formally details how predictions are made given a model built by Function BuildTree and a new data instance. These procedures emcompass a wide range of decision tree algorithms, including the popular CART (3), ID3 (48), and C4.5 (49), as well as the bipartite trees we explore in the present work (1).

The following sections provide further considerations on specific components.

2.3.1 Searching for the best split

We explore two main strategies for the FindSplit procedure: the exhaustive search and the randomized search. The exhaustive search is the most common. The randomized search is a faster but less accurate alternative, intended to be used in the context of tree ensembles (Section 2.5). We describe each procedure in the present section.

Function Predict(RootNode, x_{new}): Compute a Decision Tree's prediction.

```
Input: A new interaction sample to be evaluated and the root node of a Decision Tree.
Output: The Decision Tree's predicted value for the given sample attributes.
Node ← RootNode;
while Node is not a leaf do
    if $x_{new}^{[Node.feature]} > Node.threshold$ then
        Node ← Node.childRight
    else
        Node ← Node.childLeft
    end
end
return prototype(Node.X, Node.Y)
```

Function BuildTree(X, Y): Recursively build a Decision Tree

```
Input: The training data for the current node.
Output: Current node, with all information of subsequent splits.
$\Delta I^*, f^*, t^*$ ← FindSplit($X, Y, \tilde{m}$);
// Many stopping criteria are possible
if DecideToStop($\Delta I^*, f^*, t^*, X, Y$) then
    return NodeObject {
        isLeaf ← True
        $X_{\text{leaf}} \leftarrow X$
        $Y_{\text{leaf}} \leftarrow Y$
    };
else
    Get $X_l, Y_l, X_r, Y_r$ from $f^*$ and $t^*$ (Eq. 2.4);
    return NodeObject {
        isLeaf ← False
        childLeft ← BuildTree($X_l, Y_l$)
        childRight ← BuildTree($X_r, Y_r$)
        feature ← $f^*$
        threshold ← $t^*$
    };
end
```

The most common approach to the FindSplit procedure is to consider all possible partitions of the given input data. For a feature column $X^{[\cdot f]}$, this exhaustive evaluation of partitions can be done by sorting $X^{[\cdot f]}$ and considering a threshold t between each two consecutive values in it. Notice that any threshold value between the same two consecutive $X_{\text{sorted}}^{[\cdot f]}$ elements will result in the exact same partitioning of the training set (Equation 2.4). The common practice is thus to take the averages between each two neighboring feature values. The same procedure is then repeated for each feature column, and the overall best t^* and corresponding feature index

Function FindSplitBest(X, Y)

```
   Input: A partition of the training data in a given node.
   Output: The highest quality score $\Delta I^*$ found among all splits evaluated, with its
           corresponding feature column $f^*$ and threshold value $t^*$.
 1 Initialize $S_r$ and $S_l$ as a $|Y|_j$-sized vectors;
 2 $\Delta I^*$, $f^*$, $t^* \leftarrow \mathbf{0}$;
 3 Draw $\tilde{m}$ columns (features) of $X$ without replacement;
 4 foreach feature index $f$ of the $\tilde{m}$ drawn features do
 5     $n_l \leftarrow 0$;                                        // Holds $|Y_l|_i$
 6     $S_r \leftarrow \sum_i Y^{[i\hat{\jmath}]}$;                         // Holds $\sum_i Y_r^{[i\hat{\jmath}]}$
 7     $S_l \leftarrow \mathbf{0}$;                                       // Holds $\sum_i Y_l^{[i\hat{\jmath}]}$
 8     Get the permutation $P$ that sorts $X^{[\cdot f]}$;
 9     Apply $P$ to $Y$'s and $X^{[\cdot f]}$'s rows:
10         $Y_P$, $X_P \leftarrow P(Y)$, $P(X^{[\cdot f]})$;
11     foreach row index $\hat{\imath}$ of $Y_P$ do
12         $n_l \leftarrow n_l + 1$;
13         foreach column index $\hat{\jmath}$ of $Y_P$ do
14             $S_r^{[\hat{\jmath}]} \leftarrow S_r^{[\hat{\jmath}]} - Y_P^{[\hat{\imath}\hat{\jmath}]}$;
15             $S_l^{[\hat{\jmath}]} \leftarrow S_l^{[\hat{\jmath}]} + Y_P^{[\hat{\imath}\hat{\jmath}]}$;
16         end
17         Use $S_l$, $S_r$ and $n_l$ to calculate $\Delta I$ (Equation 2.5). Notice that other
             node-specific constants might be needed;
18         if $\Delta I > \Delta I^*$ then
19             $\Delta I^* \leftarrow \Delta I$;
20             $f^* \leftarrow f$;
21             $t^* \leftarrow \frac{1}{2}(X_P^{[\hat{\imath}]} + X_P^{[\hat{\imath}+1]})$;
22         end
23     end
24 end
25 return $\Delta I^*$, $f^*$, $t^*$;
```

f^* are selected. The exhaustive split search procedure is detailed by the Function FindSplitBest.

The randomized search is an alternative that avoids considering all partitioning options and greatly reduces the amount of operations performed. It consists of drawing a random threshold t between the minimum and maximum value of each feature, thus evaluating only $|X|_j$ splits when choosing the best (line 13). Although degrading the performance of a single tree, this procedure is an interesting option when building tree ensembles (Section 2.5), being the core idea behind the extremely randomized trees algorithm (2). Ensembles of decision trees will be discussed in Section 2.5.

```
Function FindSplitRandom($X$, $Y$)
    Input: A partition of the training data in a given node.
    Output: The highest quality score $\Delta I^*$ found among all splits evaluated, with its
            corresponding feature column $f^*$ and threshold value $t^*$.
    $\Delta I^*$, $f^*$, $t^* \leftarrow \mathbf{0}$;
    Draw $\tilde{m}$ columns (features) of $X$ without replacement;
    foreach feature index $f$ of the $\tilde{m}$ drawn features do
        Find min($X^{[\cdot f]}$) and max($X^{[\cdot f]}$);
        Draw a random threshold value $t \in \mathbb{R}$ so that min($X^{[\cdot f]}$) $< t <$max($X^{[\cdot f]}$);
        Calculate $\Delta I$ for the drawn $t$ (Equation 2.5) ;                // $O(|Y|)$
        if $\Delta I > \Delta I^*$ then
            $\Delta I^* \leftarrow \Delta I$;
            $f^* \leftarrow f$;
            $t^* \leftarrow \frac{1}{2}(X_P^{[\hat{i}]} + X_P^{[\hat{i}+1]})$;
        end
    end
    return $\Delta I^*$, $f^*$, $t^*$
```

2.3.2 Measuring the quality of a split

A split quality criterion must be defined so we can compare and select the best splitting rules at each node. The quality ΔI of a split is commonly framed as the decrease of an impurity metric calculated over the partitions of training labels (Equation 2.5). This decrease is taken for the combined impurities of the generated children nodes (Equation 2.4). All impurities are multiplied by the size of each partition relative to the total number of training samples ($|Y_{\text{root}}|$), restricting the effect of nodes with less data that could introduce spurious variations of impurity. Notice that $|Y_{\text{node}}| = |Y_{\text{left}}| + |Y_{\text{right}}|$.

$$\Delta I(Y, t, f) = \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} I(Y_{\text{node}}) - \frac{|Y_{\text{left}}|}{|Y_{\text{root}}|} I(Y_{\text{left}}) - \frac{|Y_{\text{right}}|}{|Y_{\text{root}}|} I(Y_{\text{right}}) \tag{2.5}$$

Several metrics can be chosen as the impurity function $I(\cdot)$, such as the Gini impurity, the Shannon entropy or the Poisson loss (3). In this study we utilize the variance of each output column, averaged over all outputs (Equation 2.6).

$$I_{\text{MSE}}(Y) = (Y^{[ij]} - Y^{\langle i\rangle[j]})^{2\langle ij\rangle} = Y^{2\langle ij\rangle} - Y^{\langle i\rangle 2\langle j\rangle} \tag{2.6}$$

Most commonly, the prototype function returns the column averages of a leaf's partition of the training labels: $Y_{\text{leaf}}^{\langle i\rangle[j]}$. In this case, the column variances correspond to the *mean squared error* (MSE) for the training data in the node, as if the node holding $Y_{\text{partition}}$ were to become a leaf.

Also notice that I_{MSE} is equivalent to the Gini impurity if Y contains only binary values. That can be shown by noticing that $Y_{\text{bin}}^{[ij]2} = Y_{\text{bin}}^{[ij]}$ for binary labels, so we define $\mathbf{p}$ as $\mathbf{p}^{[j]} =$

$Y_{\text{bin}}^{2\langle i\rangle[j]} = Y_{\text{bin}}^{\langle i\rangle[j]}$, which yields Equation 2.7.

$$I_{\text{MSE}}(Y_{\text{bin}}) = Y_{\text{bin}}^{2\langle ij\rangle} - Y_{\text{bin}}^{\langle i\rangle 2\langle j\rangle} = (\mathbf{p}^{[j]} - \mathbf{p}^{\langle j\rangle 2})^{\langle j\rangle} = [\mathbf{p}^{[j]}(1 - \mathbf{p}^{[j]})]^{\langle j\rangle} = I_{\text{Gini}}(Y) \tag{2.7}$$

2.4 Bipartite decision trees

Some learning algorithms can be adapted in a deeper level to the bipartite context. These algorithms are able to directly receive the two X matrices as input, without needing to combine them as a preprocessing step for training. We term them *estimator-centric* adaptations, in contrast with the data-centric strategies discussed in Section 2.2. This section explores the main focus of this work: estimator-centric adaptations of decision tree models. Appendix A presents two other estimator-centric strategies, based on linear models and matrix factorization.

Essentially, the adaptation process consists of defining bipartite versions of each of the four key components of the decision tree algorithm (Section 2.3): the FindSplit procedure, the split quality metric, the stopping criteria, and the prototype function. From these, the most central is the FindSplit procedure adaptation.

For bipartite trees, as proposed by 1, we perform the split search locally in each domain, in a similar fashion to the SLMO adaptation (Section 2.2.2). Specifically, let `FindSplit` $_{\text{trad}}$ be a procedure for finding a split threshold in a traditional multioutput decision tree (see Section 2.3.1). A bipartite tree applies `FindSplit` $_{\text{trad}}$ twice at each node: once over X_1 and Y and once over X_2 and $Y^\intercal$. This results in one split being chosen for each axis. Finally, the best overall split is selected between the vertical and horizontal splits. In essence, a bipartite decision tree algorithm uses a `FindSplit` procedure that composes two traditional `FindSplit` $_{\text{trad}}$ procedures, as detailed by Function FindBipartiteSplit.

2.4.1 Bipartite global single-output trees

This section shows how we can grow bipartite trees more efficiently, making tree estimators more scalable to large bipartite datasets.

The original bipartite tree proposed by 1 uses the impurity metric I_{MSE} (Equation 2.6) to evaluate splits, so that the authors classify their technique as a *Global MultiOutput* (GMO) estimator. *Multi-output* because the impurity evaluates the average variance of each label column (or row), as if each column (or row) were a different output. *Global* because each tree utilizes features from both feature domains for training (see Section 2.2). Their GMO trees, however, are shown to have the same algorithmic complexity as traditional decision trees trained with the SGSO adaptation. We demonstrate that faster training algorithms can be developed if we assume a single-output format under the context of bipartite trees.

The split rules as defined by Equation 2.4 are agnostic to the specific arrangement of the bipartite data. They can be seamlessly applicable to either the (X_{SGSO}, Y_{SGSO}) format employed by the SGSO adaptation (Equation 2.3) or directly to the X_1, X_2 and Y matrices. Furthermore,

```
Function FindBipartiteSplit($X$, $Y$)
    Input: A partition of the bipartite training data in a given node. $X$ encodes one
        design matrix for each axis, $X_1$ and $X_2$.
    Output: The highest quality score $\Delta I^*$ found among all splits evaluated in both
        row and column directions, with its corresponding feature column $f^*$ and
        threshold value $t^*$.
if adapterStrategy is BGSO (Section 2.4.1) then
    // Build $Y$ proxies $\tilde{Y}_1$ and $\tilde{Y}_2$ (Equation 2.10)
    $\tilde{Y}_1 \leftarrow Y^{[\cdot]\langle j\rangle}$;
    $\tilde{Y}_2 \leftarrow Y^{\langle i\rangle[\cdot]}$;
else
    // Using GMO strategy, no proxies are used
    $\tilde{Y}_1 \leftarrow Y$;
    $\tilde{Y}_2 \leftarrow Y^\intercal$;
end
// Generate a split in each axis. Get each split's
    position, feature and quality score
$\Delta I_r^*, f_r^*, t_r^* \leftarrow$ FindSplit($X_1$, $\tilde{Y}_1$);
$\Delta I_c^*, f_c^*, t_c^* \leftarrow$ FindSplit($X_2$, $\tilde{Y}_2$);
if $\Delta I_r^* > \Delta I_c^*$ then
    return $\Delta I_r^*$, $f_r^*$, $t_r^*$
else
    // $f_c^*$ value lets clear its $X_2$ ownership
    return $\Delta I_c^*$, $f_c^*$, $t_c^*$
end
```

the impurity at each node of an SGSO tree, in many cases can be translated to the bipartite format. Equation 2.8 describes such a translation of the MSE impurity presented by Equation 2.6.

$$I_{\text{MSE}}(Y_{\text{SGSO}}) = Y^{2\langle ij\rangle}_{\text{SGSO}} - Y^{\langle i\rangle 2\langle j\rangle}_{\text{SGSO}} = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle 2} \tag{2.8}$$

If we then define

$$I_{\text{GMSE}}(Y) = I_{\text{MSE}}(Y_{\text{SGSO}}) = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle 2} \tag{2.9}$$

the exact same SGSO-adapted decision tree, with its node structure and split rules, can be grown by applying the bipartite procedure (Function FindBipartiteSplit) with the I_{GMSE} impurity instead of the original I_{MSE}. This again follows from the generality of the split rules defined in Equation 2.4, invariant under the SGSO data rearrangement (Equation 2.3). Notice how the squared divergences in 2.6 are computed relative to the average of each Y column, while in 2.9 the inner average is computed over the whole Y matrix.

We can explore this property of the GSO impurity to iterate more efficiently over candidate splits. This is done by pre-computing averages of each row and column of Y_{node}. We can then iterate over one-dimensional $\tilde{Y}_{\text{node}}$ proxies (Equation 2.10) instead of the bidimensional matrix when evaluating splits (Function FindSplitBest).

$$\begin{aligned}\tilde{Y}_1^{[i]} &= Y^{[i]\langle j\rangle}\\ \tilde{Y}_2^{[j]} &= Y^{\langle i\rangle [j]}\end{aligned} \tag{2.10}$$

The same is not possible for the GMO trees, as Equation 2.6 requires storing averages for each Y column, to be squared individually. Function FindBipartiteSplit describes the adapted procedure in each case. The complexity improvements are demonstrated theoretically in Section 2.4.2 and later empirically in Section 2.9.2.

2.4.2 Asymptotic complexity analysis

In this section, we derive the asymptotic complexity of the proposed algorithms for growing bipartite trees.

We consider that the number of horizontal instances has the same magnitude as the number of vertical instances, so that we define $n \sim |X_1|_i \sim |X_2|_i$. Similarly, we assume $m \sim |X_1|_j \sim |X_2|_j$ and define $\tilde{m}(m)$ as the effective number of features. That is, the split search procedure samples $\tilde{m}(m)$ features to consider in each node. The sampling is most often done without replacement, so that $\tilde{m}(m) \in O(m)$. Common choices are $\tilde{m}(m) = \lceil\sqrt{m}\rceil$ or $\tilde{m}(m) = \lceil\log_2(m)\rceil$.

From the algorithm description, we can infer that the complexity of both Function FindSplitBest and line 13 will be given by

$$\begin{aligned}\texttt{FindSplit}(n, \tilde{m}) &\in \Theta(\tilde{m}S(|Y|_i) + \tilde{m}|Y|)\\ &= \Theta(\tilde{m}S(n) + \tilde{m}n^2)\\ &= \Theta(\tilde{m}n^2)\end{aligned} \tag{2.11}$$

where $S(n)$ is the complexity of the chosen sorting algorithm used by `FindSplit` when operating on n values. When features are real-valued, the current most effective algorithms are $\Theta(n \log n)$, as, for example, Quick Sort or Merge Sort (50). For integer features, the complexity is $\Theta(n)$, as in Counting Sort or Radix Sort (50).

When applied on X_{SGSO} and Y_{SGSO}, we still have that $|Y| = n^2$ and $\tilde{m}(m) \in O(m)$.

$$\begin{aligned}\texttt{FindSplit}_{\text{SGSO}}(n, \tilde{m}) &\in \Theta(\tilde{m}S(|Y_{\text{SGSO}}|_i) + \tilde{m}|Y_{\text{SGSO}}|)\\ &= \Theta(\tilde{m}S(n^2) + \tilde{m}n^2)\\ &= \Theta(\tilde{m}S(n^2))\end{aligned} \tag{2.12}$$

For BGSO, the complexity is given by

$$\begin{aligned}\texttt{FindSplit}_{\text{BGSO}}(n, \tilde{m}) &\in \Theta(n^2 + \tilde{m}S(|\tilde{Y}_1|_i) + \tilde{m}|\tilde{Y}_1|)\\ &= \Theta(n^2 + \tilde{m}S(n) + \tilde{m}n)\\ &= \Theta(n^2 + \tilde{m}S(n))\end{aligned} \tag{2.13}$$

The complexity of FindBipartiteSplit will be the same as the non-bipartite `FindSplit` that it wraps, since it simply applies the wrapped function twice at each node.

To estimate the complexity of the whole tree-building process, let's consider the ideal scenario of a balanced decision tree. We also consider that the axis of a split alternates at each level, so that a matrix Y is separated in four equal sized pieces after two levels. This results in the following recurrence relationship:

$$T(n) = 4T\left(\frac{n}{2}\right) + \texttt{FindSplit}(n, \tilde{m}) \tag{2.14}$$

$T(n)$ is the time to build the tree from an n by n interaction matrix, and $\texttt{FindSplit}(n, \tilde{m})$ in this case is the time taken to select a split. The algorithm complexity of such recursive functions then follows the master theorem (50). The theorem gives the time complexity of a function T obeying the recurrence relation $T(n) = aT(n/b) + F(n)$ and $c = \log_b a$:

$$T(n) \in \begin{cases} \Theta(n^c) & \text{if } F(n) \in O(n^{c-\epsilon}). \\ \Theta(n^c \log^{k+1} n) & \text{if } F(n) \in \Theta(n^c \log^k n). \\ \Theta(f(n)) & \text{if } F(n) \in \Omega(n^{c+\epsilon}) \text{ and } F(n) \text{ is regular.} \end{cases} \tag{2.15}$$

In Equation 2.15, ϵ is a positive infinitesimal constant and k is any non-negative integer. For the third case, we say a function $F(n)$ is regular if it satisfies the *regularity condition*: $aF(n/b) \leq qF(n)$, for some constant $q < 1$ and all sufficiently large n (50).

For our tree algorithms, Equation 2.14 shows that $c = 2$ and $F(n)$ represents $\texttt{FindSplit}(n, \tilde{m})$. The resulting complexities are presented by Table 1. We see that the choice of the sorting function only affects the SGSO-adapted trees. In the worst case, where $S(n) = n \log n$, the SGSO trees are $\log n$ times slower than the GMO trees. If $S(n) = n$ instead, both have the same complexity.

Table 1 also shows that BGSO trees are faster than GMO trees when considering a high number of features. Specifically, if $\tilde{m}(n) \in O(\log n)$, BGSO will be $\tilde{m}$ times faster than GMO. If $\tilde{m}(n) \in \Omega(\log n)$, BGSO will be $\log n$ times faster than GMO. The only case in which both are equivalent is when $\tilde{m} \in O(1)$.

Related experiments

- **Section 2.9.2 – Are BGSO models faster than GMO models?**

  Presents empirical evidence of the efficiency gains of the proposed algorithm, comparing BGSO against GMO decision trees.

Table 1 – Comparison between asymptotic time complexities of decision tree-building procedures. We assume $n \sim |X_1|_i \sim |X_2|_i$. Similarly, $\tilde{m}$ represents the number of features to be considered for the split search in each node. $S(n)$ denotes the complexity of the sorting procedure, usually n for integral features or $n \log n$ for decimal values. The last column refers to the case where $\tilde{m} \sim n$. This scenario could arise, for instance, when X_a are pairwise similarities or kernel matrices.

Strategy	Split search	Tree building	$\tilde{m} \propto n$
SGSO	$\Theta(\tilde{m}nS(n))$	$\Theta(\tilde{m}nS(n)\log n)$	$\Theta(n^2 S(n)\log n)$
GMO	$\Theta(\tilde{m}n^2)$	$\Theta(\tilde{m}n^2 \log n)$	$\Theta(n^3 \log n)$
BGSO	$\Theta(n^2 + \tilde{m}S(n))$	$\Theta(n^2(\log n + \tilde{m}))$	$\Theta(n^3)$

Source: By the author.

```
Function BipartiteTreePredict(RootNode, $\mathbf{x}_{1,\text{ new}}$, $\mathbf{x}_{2,\text{ new}}$)
    Input: A new dyad and the root node of a Bipartite Decision Tree.
    Output: The tree's predicted value for the given feature vectors.
1   Node ← RootNode;
2   while Node is not a leaf do
3       if Node.axis $= 1$ then $\mathbf{x}_{\text{new}} \leftarrow \mathbf{x}_{1,\text{ new}}$;
4       else $\mathbf{x}_{\text{new}} \leftarrow \mathbf{x}_{2,\text{ new}}$;
5       if $\mathbf{x}_{new}^{[Node.feature]} >$ Node.threshold then Node ← Node.childRight ;
6       else Node ← Node.childLeft;
7   end
8   return prototype($\mathbf{x}_{1,\text{ new}}$, $\mathbf{x}_{2,\text{ new}}$, Node.$Y$)
```

2.4.3 Prototype functions for bipartite trees

In this work we propose a different prototyping strategy to determine the output value of each leaf in a global multi-output (GMO) decision tree, taking the similarity matrices of our use cases into consideration.

When a leaf is reached during the prediction step of a decision tree, the `prototype` function is called to determine the output value to be returned (line 8 of Function BipartiteTreePredict). With traditional datasets, the `prototype` function most often returns the average label of the leaf's partition (Equation 2.16) (3).

$$\texttt{prototype}\,(Y_{leaf})^{[j]} = Y_{\text{leaf}}^{\langle i\rangle[j]} \tag{2.16}$$

As an extension, the most natural approach to use on single-output bipartite trees is the analogous average of the whole partition of the interaction matrix (Equation 2.17).

$$\texttt{prototype}_{\texttt{GMOsa}}(\mathbf{x}_{1,\text{ new}}, \mathbf{x}_{2,\text{ new}}, Y_{\text{leaf}}) = Y_{\text{SGSO, leaf}}^{\langle i\rangle[1]} = Y_{\text{leaf}}^{\langle ij\rangle} \tag{2.17}$$

$X_{1,\text{ leaf}}$, $X_{2,\text{ leaf}}$ and Y_{leaf} represent the partition of the training set that reaches the leaf. Nevertheless, some considerations are possible when dealing with bipartite data, since there are cases in which one of the entities of the interaction being predicted is already known from the training set. As introduced by (1), if a row instance is in the training set, we have the option of averaging only the column of Y_leaf corresponding to its known outputs (the analogous being performed for a known column instance). Specifically, when predicting the interaction between a sample pair $\mathbf{x}_{1,\text{ new}}$ and $\mathbf{x}_{2,\text{ new}}$, we can set `prototype` as in Equation 2.18 (the arguments are omitted).

$$\texttt{prototype}_\texttt{uniform}(\cdot) = \begin{cases} Y_\text{leaf}^{[k]\langle j\rangle} & \text{if } \exists\ k \mid \mathbf{x}_{1,\text{ new}} = X_{1,\text{ leaf}}^{[k]} \\ Y_\text{leaf}^{\langle i\rangle[k]} & \text{if } \exists\ k \mid \mathbf{x}_{2,\text{ new}} = X_{2,\text{ leaf}}^{[k]} \\ Y_\text{leaf}^{\langle ij\rangle} & \text{otherwise.} \end{cases} \tag{2.18}$$

A drawback of this approach is a possibly greater susceptibility to random fluctuations, since the label averages in the prediction step are taken over a much smaller sample size (a single row or column of Y_leaf instead of the whole Y_leaf). Given we are working with similarity scores, we propose an intermediate approach. We weight the rows and columns of Y_leaf by similarity values in the form $\mathbf{s}_a^{[i]} = \text{similarity}(\mathbf{x}_{a,\text{ new}}, X_a^{[i]})$ between $\mathbf{x}_\text{new}$ and the training samples (Equation 2.19).

$$\texttt{prototype}_\texttt{weighted}(\cdot) = \frac{\sum_{i\in\text{leaf}} \mathbf{s}_1^{[i]} Y_\text{leaf}^{[i]\langle j\rangle}}{2\sum_{i\in\text{leaf}} \mathbf{s}_1^{[i]}} + \frac{\sum_{j\in\text{leaf}} \mathbf{s}_2^{[j]} Y_\text{leaf}^{\langle i\rangle[j]}}{2\sum_{j\in\text{leaf}} \mathbf{s}_2^{[j]}} \tag{2.19}$$

Furthermore, the similarities we are dealing with are precomputed: X_1 and X_2 are square matrices in which $X_a^{[i_1 i_2]} = \text{similarity}(X_a^{[i_1]}, X_a^{[i_2]})$. Therefore, $\mathbf{x}_{a,\text{ new}}$ already represents a similarity vector $\mathbf{s}_a$. We explore this property in three different ways, termed `precomputed`, `square`, and `softmax`:

$$\mathbf{s}_{a,\texttt{precomputed}} = \mathbf{x}_{a,\text{ new}} \qquad \mathbf{s}_{a,\texttt{square}}^{[i]} = (\mathbf{x}_{a,\text{ new}}^{[i]})^2 \qquad \mathbf{s}_{a,\texttt{softmax}}^{[i]} = \exp(\mathbf{x}_{a,\text{ new}}^{[i]}) \tag{2.20}$$

The first case uses the similarity scores directly. The second case employs the square of the similarity scores to emphasize the importance of the most similar samples. The third case amplifies even further this effect by using the exponential of the similarity scores. Experimental comparisons between these three prototype functions are presented in Section 2.9.3.

Related experiments

- **Section 2.9.3 – Which prototype should a GMO forest use?**

  Compares the performance of the three prototype functions in the context of bipartite GMO decision forests.

2.5 Decision forests

While single decision trees are valuable tools for comprehending the learning problem at hand, they often fall short in effectively modeling intricate relationships within the data, displaying limited generalization capabilities and high susceptibility to overfitting (32, 33). Their most significant impact on machine learning applications is observed when committees of such models are utilized, in which the final output values are, for example, the average predictions of all trees. These compositions of estimators are usually referred to as *ensembles* (32,51,52). Being studied in the context of machine learning since the 1970s (51,52), they reflect the intuitive idea that combining multiple opinions from a diverse set of experts frequently results in better decision taking.

In fact, it has been extensively demonstrated both empirically and theoretically that the predictive perfomance a group of learners always surpasses that of its individual components if and only if the individual estimators are sufficiently accurate and diverse (32, 34, 51, 52). Importantly, requiring diversity means that the individual estimators must ideally commit errors on different instances for the composition to succeed (52). 51,52 describe three ways in which combining diverse estimators could benefit the ensemble's performance:

1. **Statistical:** Building each estimator can be seen as finding a hypothesis that explains the prediction problem as well as possible. If multiple different hypotheses are found, the likelihood that at least one of them is close to the true underlying function is increased. Furthermore, averaging multiple hypotheses can contribute to alleviate the influence of each hypothesis' variance, resulting in a more accurate approximation and reduced propensity to overfitting.

2. **Computational:** Finding the globally optimal decision tree is known to be an NP-complete problem (53). Thus, finding and combining multiple approximate hypotheses, that may represent local maxima of the objetive function, is often much cheaper than expending time on the search for a global solution.

3. **Representational:** Sometimes, a single estimator is not complex enough to represent the intricacies of the probem at hand. However, combination of models can be utilized to enhance the representational power of the ensemble, building a more general decision function as a combination of the simpler decision boundaries of each estimator.

Being diversity a key factor, mechanisms of introducing model heterogeneity are as important to consider as strategies for enhancing the base algorithm. In fact, many diversity-inducing techniques even degrade the performance of individual estimators, but still result in improvements for the ensemble as a whole (2,32). For example, one of the most straighforward ways to promote heterogeneity is to build each estimator on randomized subsets of the training data. In this and in many other cases, designing ensemble models is thus a matter of balancing diversity and individual strength (32).

This is the core idea of *decision forests* (the ensembles of decision trees). Even simple estimators such as decision trees, traditionally prone to overfitting, can be leveraged to com-pose powerful learning algorithms. We briefly describe some popular strategies for introducing heterogeneity in the decision tree growing procedure:

1. **Instance sampling:** A new version of the training set is built by randomly drawing instances from it, with or without replacement (selecting rows of X). If drawing with replacement, one can optionally draw the same number of instances as the original set, a procedure known as bootstrapping.
2. **Feature sampling:** A random subset of features is selected to be used (selecting columns of X). The number of features $\tilde{m}$ selected in each node is commonly defined as a function of the total number of features $|X|_j$. Usual choices are $\tilde{m} = \lceil \sqrt{|X|_j} \rceil$ or $\tilde{m} = \lceil \log |X|_j \rceil$.
3. **Split threshold randomization:** Instead of searching for the best split threshold for each feature as in Function FindSplitBest, a random threshold value is drawn between the minimum and maximum values of each feature column. This procedure is described in line 13.

Both instance and feature undersampling can be performed node-wise, occurring before the split search procedure of each node, or tree-wise, occurring once for each tree in the ensemble. Both can also be performed with or without replacement, but notice that sampling features with replacement only makes sense if randomization of split threshold is also employed. In such a case, a random candidate split threshold is selected for each repetition of a feature, whereas the same split threshold would be selected for all duplicates if the greedy approach was used, spending more time with no different result than if omitting the repeated features.

The concept of sampling instances with replacement to create a different training set for each estimator in the ensemble was first proposed by 54 under the name of bootstrap aggregation or *bagging*. The idea of selecting a subset of features was introduced independently by 55 and 56. 55 explored feature sampling as a remedy for a shape recognition problem with impeditively large sample sets, whereas 56 was mainly focused on overfitting-prevention for decision forests. It was 32 who first combined the two ideas, proposing and greatly popularizing the Random Forest algorithm, with over 120 thousand citations according to Google Scholar as of March 2024. 2 later introduced the randomized split threshold concept, presenting the Extremely Randomized Trees algorithm (Extra-Trees). The algorithm shows competitive prediction scores and clear superiority in terms of training speed compared to bagging and Random Forests (2, 17).

The two main ensemble-building strategies we explore in the current work can now be defined as follows:

- **Random Forests (RF) (32):** Tree-wise instance sampling with replacement and node-wise feature undersampling without replacement are employed.

- **Extremely Randomized Trees (ERT) (2):** Split threshold randomization and node-wise feature undersampling without replacement are employed. No resampling of instances is performed in the original proposal.

We refer the reader to 57 and 58 for a more in-depth description of prominent decision forest strategies and previous work in the field. 59 provides an experimental comparison of the most popular ensemble methods. Regarding interaction problems, 17 explores the use of decision forests under the standard global multi-output and local single-output adaptations presented by Section 2.2.

The same tree-diversification and forest building techniques discussed in this section can also be applied to bipartite decision trees, with very small modifications regarding the data sampling procedures: instance sampling and feature sampling must now occurr on both domains of the interaction dataset. 18 explores these ideas, using bipartite global multi-output decision trees to build both Random Forests and Extra-Trees ensembles. We hereafter refer to these forests as Bipartite Random Forests (BRF) and Bipartite Extra-Trees (BXT), respectively. Their study suggests superior performance of BXT in comparison to SLMO- and SGSO-adapted forests, as well as in comparison to previously proposed algorithms (18). The authors later extend their work to include a label imputation step for predicting drug-target interactions (29). In that study, their model is built on a reconstructed version of the interaction matrix obtained through Neighborhood-Regularized Logistic Matrix Factorization (NRLMF) (31). The NRLMF algorithm is described in detail in Section A.2.

Related experiments

- **Section 2.9.4 – Which adaptation strategy is the best for decision forests?**

  Compares the performance of different bipartite decision forests. Both data-centric (Section 2.2) and estimator-centric (Section 2.4) adaptations are explored.

- **Section 2.9.5 – Can label imputation assist bipartite forests?**

  Investigates the impact of NRLMF as a method to impute positive annotations. All bipartite adaptations are tested on the reconstructed matrix.

2.6 Incorporating semi-supervision into decision trees

The tree algorithms presented up until this point do not consider the positive-unlabeled (PU) property of interaction data. They work under the assumption of a *supervised* scenario, in which all annotations are reliable. In this section, we discuss adaptations to decision trees specifically designed to the challenges of PU data.

As several previous authors (19, 60, 61), we argue that accounting for the partial availability of information could improve prediction performance of bipartite models. The strategies to deal with missing labels are the main concern of the *semi-supervised* learning paradigm (20, 41, 62), which we briefly introduce.

2.6.1 Semi-supervised learning

Supervised problems are those we have been exploring, in which the goal is to build a model that can predict the label of a new instance based on its features (63, p. 9). We have a matrix Y of target values and feature matrices X, and the model's objective is to learn the relationship between X and Y.

When the learning problem at hand does not involve predefined labels, the task is termed *unsupervised* (63, p. 486). The most common example is clustering problems, where the goal is to group similar instances together based solely on their numerical attributes. In these problems, there is not a preconceived target to model, such as desired output values or classes to which each instance should belong. Only the X matrix is utilized, there is not an associated label matrix Y.

In summary, while supervised learning problems are concerned with the relationship between X and Y, unsupervised learning problems target the relationships between instances themselves. While supervised learning focuses on the graph of a function $f : \mathbf{x} \rightarrow \mathbf{y}$, unsupervised learning focuses on the structure of the feature space.

Some problems, however, lie in the intersection of the supervised-unsupervised spectrum. They usually arise when the available annotations are incomplete or unreliable, so that both supervised and unsupervised objectives are of interest. Take for instance the case of an image gallery application capable of grouping all photos of a person into a folder with their name. While the names must be asked once to the user, all photos of each different person are already grouped beforehand. The algorithm is both concerned with labeling each photo with the correct name and with grouping similar photos together. Another example can occur when building a medical diagnosis database. Such a database could be a result of merging multiple datasets and considering a large collection of patients. Some amount of missing information is very common in these cases, and the algorithms must be able to deal with these vacancies. One possible strategy is to infer missing labels from the labels of similar instances, bringing the concept of clustering again into play.

In the examples, both label inference (that characterizes supervised problems) and clustering (that characterizes unsupervised problems) are intertwined as the model's objective. The goal is to simultaneously learn to correctly classify the instances with known labels while also considering similarities to infer the missing or uncertain annotations. The class of such hybrid learning tasks is called *semi-supervised learning* (20, 62).

Since missing labels are a defining characteristic of interaction prediction tasks, it is commonly suggested (31,43,61) that applying semi-supervised concepts to our problems could significantly improve the performance of bipartite models. We then explore this hypothesis in the context of bipartite decision trees.

2.6.2 Semi-supervised decision trees

This section describes our strategies to incorporate semi-supervision assumptions into bipartite decision trees.

To develop semi-supervised algorithms, assumptions must be made about the data. The main assumption governing our approach is that similar instances are likely to have similar labels. This is known as the *smoothness* assumption (20, 41), and can be incorporated into decision trees by redefining the impurity function guiding the split search (39,40,43).

Notice that the growing of decision trees unavoidably induces groupings of samples in the training set. Specifically, the structure of each tree represents a hierarchical clustering of the training samples, in which each tree node represents a partition of the training set composed by the training instances that reach that node. However, this clustering procedure is usually preformed under the objective of grouping instances with similar *labels*, which not necessarily means that instances in the same group will have similar *features*. This results from the definition of the impurity function I governing the split search procedure: I is usually chosen as to minimize the divergence of labels within each partition (see Section 2.3). As such, I commonly depends on Y_{node} alone. With that in mind, decision trees can be naturally adapted to unsupervised or semi-supervised tasks by redefining the impurity function to consider the feature matrix X_{node} instead of only the label matrix Y_{node}.

In a purely unsupervised context, an example would be to utilize the average of column variances in X_{node} as the impurity function (Equation 2.21). This would result in a tree that groups instances with similar features together, regardless of their labels.

$$I_{\text{u, MSE}}(X) = \left(X^{[ij]} - X^{\langle i \rangle [j]}\right)^{\langle i \rangle 2 \langle j \rangle} \tag{2.21}$$

To address semi-supervised scenarios, we can consider both supervised and unsupervised objectives simultaneously. This can be done by using a linear combination of unsupervised and supervised impurities to guide the split selection, taking into account both the similarities between features and between labels to build a semi-supervised decision tree (39,40,43). Equation 2.22 defines such a hybrid impurity function.

$$I_{ss}(X_{\text{node}}, Y_{\text{node}}) = (1-\sigma)\frac{I_u(X_{\text{node}})}{I_u(X_{\text{root}})} + \sigma\frac{I_s(Y_{\text{node}})}{I_s(Y_{\text{root}})} \tag{2.22}$$

We divide each term by the corresponding impurities on the root node, thus calculated over the entire training set. The reason is to avoid the influence of possible differences in scale between the values in X and in Y. It also compensates relative scale differences that could arise when choosing different functions for I_u and I_s. The parameter $\sigma \in [0, 1]$, that we call supervision balance, weights the relative importance given to each objective, with $\sigma = 0$ corresponding to a fully unsupervised tree and $\sigma = 1$ to a fully supervised tree. Strategies for adjusting σ are discussed in Section 2.6.4.

For semi-supervised tasks in general, we usually have confirmed positive and confirmed negative annotations alongside missing entries. In this case, notice how I_s can only be calculated over the non-missing annotations, while I_u can always utilize both labeled and unlabeled instances. Nevertheless, all confirmed annotations are positive in our present scenario of positive-unlabeled learning. Thus, we must still consider the missing labels as zero entries for the calculation of I_s. Even so, we argue that decreasing the relative importance of the supervised objective can compensate the label uncertainty and improve generalization. The idea is to encourage the tree to appreciate the structures in the feature space. For example, it should select splits that delineate very compact groups of instances, even if the labels in a given group are not satisfactorily homogeneous.

A caveat of using a semi-supervised impurity as in Equation 2.22 is that trees could continue to find splits even if all instances in a given node have the same label. This is because we could keep reducing the unsupervised impurity by further splitting even if the supervised impurity is already zero. However, the output value of each node is still calculated over Y_{node} alone, so all nodes descending from a homogeneous node would yield the same output. We avoid such redundant splits by forcefully stopping the split search for a node when $I_s(Y_{\text{node}}) = 0$.

Another important notice is that features must be normalized before training the tree: the impurity function I_u (Equation 2.21) is most often sensitive to the relative scale of the different columns of X. This is normally not a requirement for decision trees, since the split search procedure is performed separately for each feature and usually only depends on the order of the values, not on their specific magnitude.

2.6.3 Unsupervised impurities

We explore two different unsupervised impurity functions for the semi-supervised decision trees: the mean feature variance (Equation 2.21) and the mean pairwise distance between the samples (Equation 2.23). We describe and motivate them in this section.

Initial proposals of a semi-supervised impurity function (39) employed the mean variance of the feature matrix X as the unsupervised term (Equation 2.21), being concerned with traditional decision trees. The strategy, however, does not scale well with the number of features. This results from each node having to always consider the same number of features, even though the number of instances decrease. The scalability is especially a concern in the scenarios under study, in which the feature matrices are square similarity matrices. Because

X already represents similarities between samples, an additional metric such as the variance of each feature is not necessary to capture how close the samples are to each other. Exploring this property, we propose a more efficient unsupervised impurity function based on the average similarity between the samples in the tree node (equation 2.23).

$$I_{\text{MeanDistance}}(X) = \frac{1}{|\mathrm{s}_{\text{node}}|} \sum_{j \in \mathrm{s}_{\text{node}}} \sum_{i \in \mathrm{s}_{\text{node}}} (1 - X^{[ij]}) \tag{2.23}$$

s_{node} denotes the set of indices representing the samples in the node. Notice that the number of features to consider equals the number of samples in the node: we always consider a square partition of X. As such, the number of operations required by I_u drops more steeply with respect to the node size in comparison to $I_{\text{MSE GMO}}$ or similar impurities, which is especially beneficial in our case of large X matrices. We can reduce even further the number of operations by considering only the upper or lower triangle of X, exploiting X's symmetry.

We note that the subset of features is only used to calculate the impurity function, and not to select split points. The split search procedure is still performed considering all features as usual.

2.6.4 Heuristics for balancing the supervised and unsupervised objectives

Determining the ideal value of σ is not a trivial task. In fact, it is not even clear whether a single constant σ for the whole tree is enough or it should be adjusted for each node. We explore four different strategies for setting σ, that we present in this section.

43 argues that the unsupervised impurity should be more important for nodes with a larger number of unlabeled instances, since the supervised information would be more likely to be unreliable. Thus, they propose updating σ according to the label density of each node, as defined by Equation 2.24.

$$\sigma_{\texttt{density}} = 0.1 + 0.9 \cdot Y_{\text{node}}^{\langle ij \rangle} \tag{2.24}$$

However, we must recall that the prototype value of each node is only dependent on its partition of the label matrix. Therefore, if Y_{node} is close to being homogeneous, a new split is unable to cause a big overall change in the outputs for the instances involved. Specifically, the most drastic prototype change possible occurs if we perfectly separate the instances with positive labels from the instances with negative labels. When the node partition is very imbalanced, even this ideal separation will only greatly affect the very few instances with the minority label. Thus, using the heuristic of Equation 2.24 prioritizes the unsupervised objective only in cases where no significant change in the output values is expected. Additionally, few new splits are possible in near-homogeneous nodes, and we are more likely to achieve label homogeneity (and thus stop the split search) right after we achieve the cases in which I_u is highly prioritized. In summary, the heuristic of Equation 2.24 is likely to undermine the effect of the unsupervised objective.

To test this hypothesis, we propose another heuristic for σ that prioritizes the unsupervised objective in the earliest stages of the tree growing process and the supervised objective in nodes closer to the leaves (Equation 2.25).

$$\sigma_{\texttt{size}} = 1 - \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} \tag{2.25}$$

This way, we take advantage of the unsupervised impurity prior to when the label partitions are already homogeneous. Essentially, we start by performing clustering in the feature space, and then gradually move towards separating instances based on their labels. This process should be similar to a two step procedure: we first identify large structures in the feature space (when a large number of instances is still available) and then apply the label clustering separately to each of these structures. The difference is that each of our semi-supervised trees represents a gradual transition between the two steps.

We also speculate that the observed benefit of a dynamic σ could emerge from the diversity it promotes in tree learners. As discussed in Section 2.5, the diversity of the individual estimators is a key factor for the success of ensemble models. In this sense, ensuring variety of the σ parameter could even be more important than estimating its "correct" value for each partition. Therefore, we also evaluate the strategy of selecting a random σ at each node, drawn from a uniform distribution in the interval $[0, 1]$ (Equation 2.26).

$$\sigma_{\texttt{random}} \sim \mathcal{U}(0, 1) \tag{2.26}$$

The effectiveness of each strategy for determining σ is tested in Section 2.9.6.

Related experiments

- **Section 2.9.6 – What is the best way of building semi-supervised forests?**

  Presents a comparison between semi-supervised BGSO BRT with different σ heuristics and unsupervised impurities.

- **Section 2.9.7 – Which strategies are the most promising?**

  Provides a comparison between the most prominent bipartite forests investigated, including semi-supervised forests, supervised forests and forests using NRLMF for label imputation.

2.7 Assessing the performance of bipartite models

In this section, we discuss the evaluation procedure of estimators in bipartite learning settings. The positive-unlabeled (44) property of such datasets and the presence of two instance domains pose special considerations on how we assess the performance of our models. We thus present the different ways in which bipartite datasets can be split between training and test sets (Section 2.7.1), followed by a description on how cross-validation can be adapted to our settings (Section 2.7.2). Finally, we present and theoretically compare the two evaluation metrics utilized (Section 2.7.3).

2.7.1 There are multiple ways to measure model generality

To evaluate machine learning models, the standard procedure consists of separating a subset of data samples not to be used in the training process. These samples are subsequently inputted to the trained model and its known labels are compared to the model's predictions in order to estimate the algorithm performance. The held-out samples are collectively called the *test set* while the ones used for model building are called the *training set*.

Having two distinct sample groups in bipartite learning settings makes the concept of a held-out set more nuanced. We list five reasonable test configurations we encounter in the literature. Given two samples $\mathbf{x}_1$ and $\mathbf{x}_2$ from each respective domain, a "test set" could refer to:

1. **Test $\mathbf{x}_1$, test $\mathbf{x}_2$ (TT):** both $\mathbf{x}_1$ and $\mathbf{x}_2$ are not present in the training set, the model has never seen either of them before;
2. **Learned $\mathbf{x}_1$, test $\mathbf{x}_2$ (LT):** $\mathbf{x}_1$ is part of the training feature matrix $X_{1,\,\text{train}}$, but $\mathbf{x}_2$ was never seen before;
3. **Test $\mathbf{x}_1$, learned $\mathbf{x}_2$ (TL):** $\mathbf{x}_2$ is part of the training feature matrix $X_{2,\,\text{train}}$, but $\mathbf{x}_1$ was never seen before;
4. **Learned $\mathbf{x}_1$, learned $\mathbf{x}_2$, masked label (LL-M):** Some of the positive annotations in the training set are randomly masked (replacing 1 by 0). The model is trained on the masked dataset and evaluated based on the predictions for the masked positives and the negative annotations.
5. **Learned $\mathbf{x}_1$, learned $\mathbf{x}_2$, unknown label (LL-U):** Models trained with the SGSO adaptation (Section 2.2) can exclude specific labels from the training set. All instances are still used for training, but some combinations of them are not presented to the model. This corresponds to applying standard validation procedures directly on the X_{SGSO} and Y_{SGSO} matrices (Equation 2.3).

Accordingly, we sometimes call the training set the *LL set*. LL, TL, LT, and TT were already presented with similar names in previous work (1, 17, 29). The authors call them $L_r \times L_c$, $T_r \times L_c$, and so on. (11,19,31) denotes LL-U, LT, TL and TT as S1, S2, S3 and S4, respectively. (31) uses a similar notation, but uses S1 to designate LL-M instead of LL-U. The LL-U test set is most frequently used when building SGSO models (19). Using LL-M is more common for matrix factorization methods (31, 64, 65).

Note that a model that performs well on TT is likely to also perform reasonably well on the other test sets, but the opposite is not necessarily true. For instance, performing well on the TL or LT sets does not mean a model has learned information about both domains. As an example, consider a drug-protein scenario where the model is tested with new drugs and known proteins. Given a specific known protein, a model might correctly infer interactions based on its knowledge that similar drugs have interacted with that protein, or that a specific drug trait correlates well with the protein's interactome. No knowledge about the protein's features is required for the model to make such predictions. Therefore, there is no guarantee that the model will also perform well for new proteins.

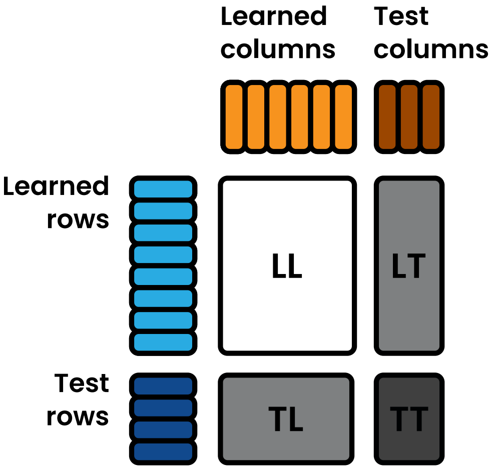

Figure 5 – Different split configurations for validating models with bipartite datasets.
Source: By the author.

Similarly, an effective model according to the LL-U or LL-M settings will not necessarily perform well on any of TT, LT or TL sets. The reason is: an estimator would still be able to perform significantly well on LL-U or LL-M using only the information from the interaction matrix (as matrix factorization algorithms do). There is no guarantee that the model will be a representation of the underlying function: $(\mathbf{x}_1, \mathbf{x}_2) \mapsto y$. As such, there is no guarantee that new instances will be correctly classified, since the only information available about them would be their feature vectors.

On the other hand, the TT test sets are the most informative in terms of generalization. They require the model to fully rely on both input vectors to predict labels, providing a more reliable representation of how attributes from both objects come together to determine the interaction. TT is therefore especially useful when one is interested in gaining insight about the underlying process that generates the interactions. However, they are also the most challenging to predict, and the model's performance on them is expected to be lower than on the other test sets.

In summary, TT should be preferred when the main interest is modeling how input features determine the interaction (called *inductive* learning (20)). TL and LT are useful if generalization is only interesting for one of the domains. LL-U and LL-M are useful when the main goal is to obtain predictions for a defined set of instances, regardless of modeling the function $(\mathbf{x}_1, \mathbf{x}_2) \mapsto y$ or not. This is called *transductive* learning (20).

For the learning problems under study, explainability is an important factor to be explored in the future. We thus report our results in terms of TT, LT and TL test sets. Additionally, the learning tasks that we approach represent very diverse phenomena (Section 2.8.1), which makes the specific LT or TL sets not comparable between datasets. We thus average both LT

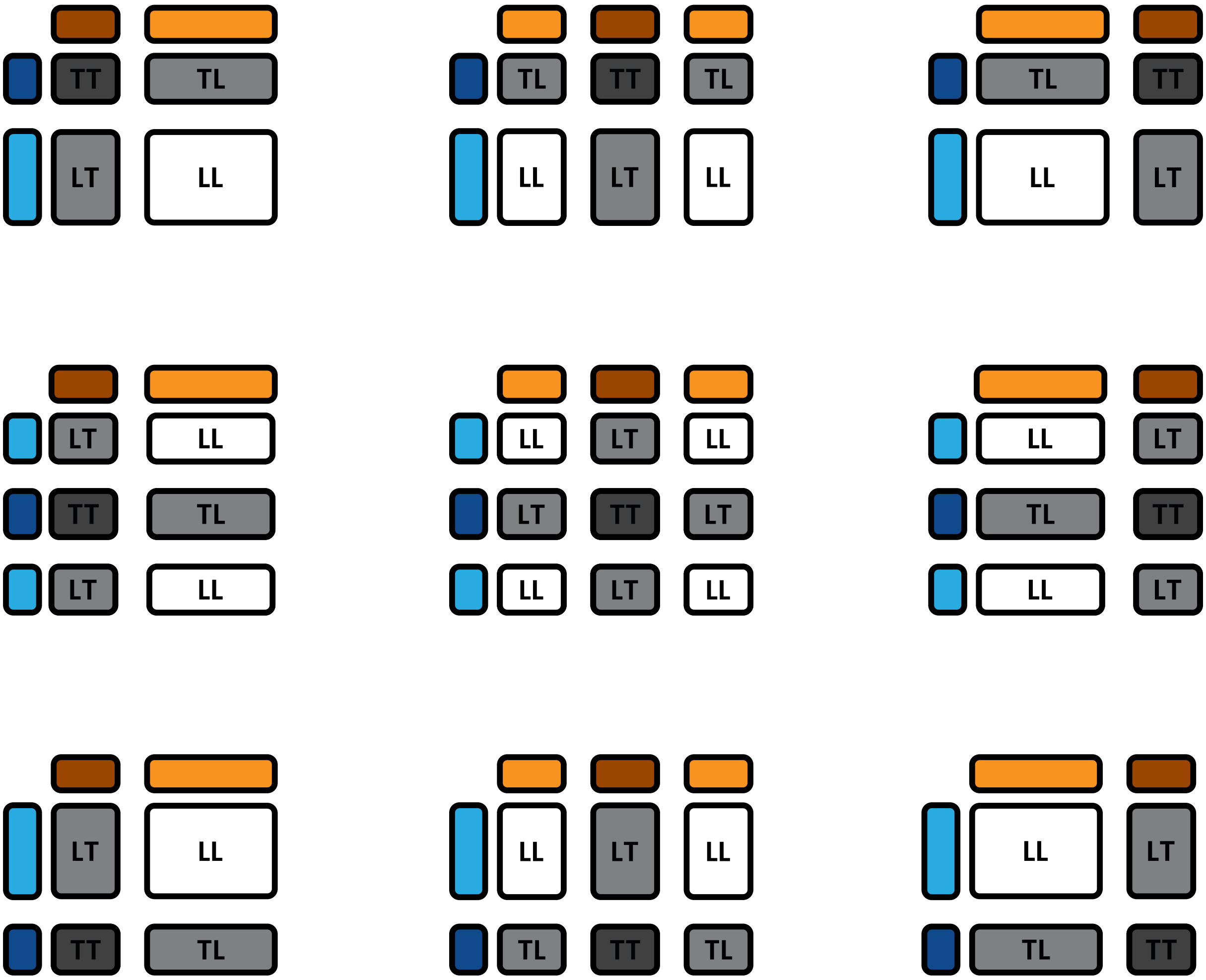

Figure 6 – Bipartite cross-validation. We represent the 9 train-test splits of a 3 by 3 bipartite cross-validation procedure.
Source: By the author.

and TL results together to report a single LT+TL score for each dataset.

2.7.2 Cross-validating in two dimensions

In traditional learning tasks, k-fold cross-validation consists in equally and randomly dividing both X and Y together in k non-overlapping partitions (or folds). The model is then evaluated k times, in each round selecting a fold as the test set and the remaining ones together as the training set.

In the bipartite interaction setting, fold division can be done in each of the two axis of the interaction matrix, corresponding to each of the two X_a sample domains $(19, 45)$. Each of the k_1 folds of X_1 can be combined with one of the k_2 folds of X_2, resulting in a corresponding partition of Y. Each bidimensional fold can be used as the TT set in a CV round, which yields the corresponding four LL, LT, TL and TT sets for each round (Figure 7). A k_1 by k_2 bidimensional CV then has a total of $k_1 k_2$ folds.

We can optionally enforce independent test sets, so that each dyad composes only one

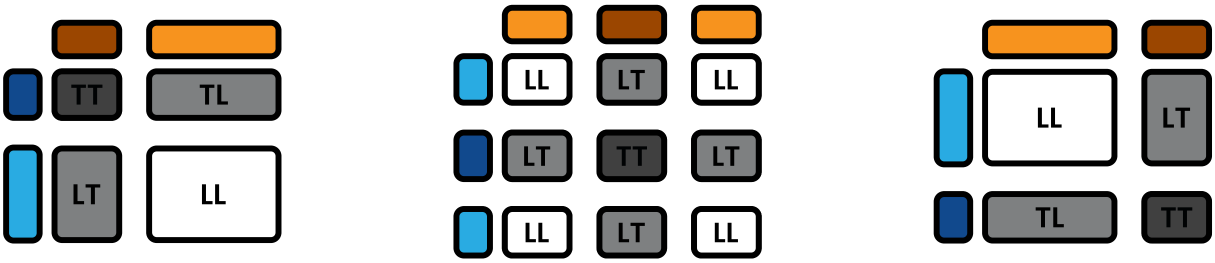

Figure 7 – Diagonal bipartite cross-validation. We represent the 3 train-test splits of a 3 by 3 diagonal bipartite cross-validation procedure.
Source: By the author.

bidimensional fold. This can be done by selecting $k = k_1 = k_2$ and pairing each X_1 partition with a single X_2 partition, yielding a total of k folds, not k^2 as before. We refer to this procedure as *diagonal* cross-validation (Figure 7).

In our settings, we average the results of all folds of each dataset before computing test statistics. Therefore, the number of folds and the independence between test folds will not affect the final significance estimates. We thus use all k^2 folds in our experiments. Diagonal cross-validation is still used, but for the nested validation used for parameter tuning in some experiments.

An important note is that the similarity matrices must be thoughtfully handled when splitting the datasets: we must not include similarities with the test samples in the training set.

2.7.3 Prediction scoring metrics

This section is concerned with defining the two metrics used throughout this work to evaluate and compare the predictive performance of estimators. The metrics are the *area under the receiver operating characteristic curve* (AUROC) and the *area under the precision-recall curve* (AUPR).

Consider a test set of N interaction labels to be inferred by a classifier (we are not concerned with the shape of Y in this section, and $N = |Y|$). Let the classifier's predictions then be represented by a matrix $\hat{Y}$ of the same shape as Y, with $\hat{Y}^{[ij]}$ being the predicted value for the ground-truth label $Y^{[ij]}$. Since Y and $\hat{Y}$ are both binary matrices, there are four possible outcomes when a prediction is made, traditionally quantified (66) as follows:

- **True Positives (TP):** the number of positive labels correctly predicted, where both the predicted and actual labels are 1.

$$TP = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 1 \text{ and } \hat{Y}^{[ij]} = 1) \tag{2.27}$$

- **True Negatives (TN):** the number of negative labels correctly predicted, where both the predicted and actual labels are 0.

$$TN = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 0 \text{ and } \hat{Y}^{[ij]} = 0) \tag{2.28}$$

- **False Positives (FP):** the number of instances where the predicted label is positive (1), but the actual label is negative (0).

$$FP = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 0 \text{ and } \hat{Y}^{[ij]} = 1) \tag{2.29}$$

- **False Negatives (FN):** the number of instances where the predicted label is negative (0), but the actual label is positive (1).

$$FN = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 1 \text{ and } \hat{Y}^{[ij]} = 0) \tag{2.30}$$

where $\mathbb{I}(A)$ is the indicator function that equals 1 if statement A is true and 0 otherwise:

$$\mathbb{I} = \begin{cases} 1 & \text{if } A \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

Notice that the sum of TP, TN, FP and FN is equal to the total number of instances T, and we also define P $= TP + FN$, the total number of *a priori* positive labels in the test set, and N $=$ $TN + FP$, the total number of *a priori* negative labels. The total number of predicted positives is termed PP $= TP + FP$ and the remaining predicted negatives are called PN $= TN + FN$. Naturally, one wants their estimator to maximize TP and TN while minimizing FP and FN.

Most commonly, we do not use these metrics directly, but instead normalize them to the interval [0, 1] in numerous ways. This enables score comparisons across datasets with different numbers of samples and different densities of positive annotations. Below we list the most common normalized scoring metrics for binary classification problems, from which the metrics used in this work are derived (66):

- **True positive rate (TPR)** or **recall:** the ratio of correctly predicted positive labels to the total number of positive labels.

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.32}$$

- **True negative rate (TNR):** the ratio of correctly predicted negative labels to the total number of negative labels.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{2.33}$$

- **False positive rate (FPR):** the ratio of incorrectly predicted positive labels to the total number of negative labels.

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR \tag{2.34}$$

- **False negative rate (FNR):** the ratio of incorrectly predicted negative labels to the total number of positive labels.

$$FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR \tag{2.35}$$

- **Precision:** the ratio of correctly predicted positive labels to the total number of predicted positive labels.

$$Pr = \frac{TP}{PP} = \frac{TP}{TP + FP} \tag{2.36}$$

An optimal binary classifier will thus present high TPR, TNR and precision while minimizing FPR and FNR.

Notice that each of these metrics still allows trivial solutions: if a classifier outputs positive labels for all instances irrespectively of the input features, it will achieve perfect TPR, FNR and Pr, but its TNR and FNR will be null. On the other hand, if a negative label is outputted every time, the opposite will happen. Thus, we will consider pairs of these metrics simultaneously, such as TPR and TNR. The selected pair must include at least three of TP, TN, FP, or FN to ensure the whole confusion matrix is taken into account.

Minimizing all the four components of the confusion matrix is not always possible. Most often, the learning algorithms are subject to a tradeoff between the ability to correctly infer positive annotations and the ability to correctly infer negative annotations, that we express as a balance between TPR and TNR. When evaluating models, we must be aware of the relative importances being assigned to each of these tendencies. This choice is highly application-specific. For instance, in the process of diagnosing medical conditions, TPR is likely to be prioritized, favoring the identification of all true cases at the expense of some misleading positive results. In this case, the cost of missing a positive result is usually far greater than that of a false positive. In other scenarios such as spam email filtering, TNR might be favored, minimizing the number of legitimate emails marked as spam even if some spam emails go undetected.

Considering this balance across a variety of learning tasks is a challenging factor to be taken into account, especially in cases where the estimators can be easily adjusted to favor each class. Many estimators, such as those employed in the present study, do not directly output a binary label but instead provide us with a continuous decision value (such as a probability of interaction) that additionally depends on a threshold parameter to establish the final predicted classes. Formally, the predicted labels $\hat{Y}$ are obtained from a threshold t applied to the decision values $\tilde{Y}$:

$$\hat{Y}^{[ij]} = \mathbb{I}(\tilde{Y}^{[ij]} > t) \tag{2.37}$$

The selection of t directly affects the propensity to positive or negative outputs, so that a single model can yield multiple different results with varying levels of TPR and TNR depending on the chosen thresholds. A common practice is then to consider TPR and TNR for all t, avoiding the influence of threshold selection on the estimator comparisons.

Notice that, since a finite set of outputs is considered for evaluation (the test set), a finite set of thresholds will cover all possible classification results of a model. These results can be easily displayed in a two-dimensional plot, using a point for each considered threshold so that its corresponding TPR and TNR are each indicated by an axis. Conventionally, the TPR is plotted in the y axis while the FPR values (representing the TNR, $FPR = 1 - TNR$) are presented as x coordinates, which results in the traditional *receiver operating characteristic* (ROC) curve $(11, 21, 22, 25)$. An ideal threshold of an ideal estimator would then be close to the top-left corner of the plot, where TPR and TNR are both 1. On the other hand, consider a completely random classifier outputting uniformly random values $\hat{Y}^{[ij]}_{\text{proba}}$ in the 0-1 interval. We would have $\hat{Y}^{[ij]} = \mathbb{I}(\hat{Y}^{[ij]}_{\text{proba}} > t)$ for a given threshold t. This implies that the number of correctly guessed positive labels is $1 - t$ (the probability of yielding 1) times the total number of positive labels: $TP = (1-t)(TP+FN)$, which results in $TPR = 1-t$ from the definition. Similarly, $FPR = TPR = 1 - t$, so that the ROC curve of a random classifier is a diagonal line from the bottom-left to the top-right corners.

To summarize a classifier's performance across all thresholds, the area under the ROC curve (AUROC) is often employed, with values ranging from 0.5 to 1, where 1 represents a perfect classifier and 0.5 represents a random classifier. Although theoretically possible, values below 0.5 would signify the opposite label is being consistently predicted, most likely indicating misconfiguration of the estimator. If AUROC < 0.5, the result can be easily converted to a value greater than 0.5 by simply inverting the predicted labels (turning 0s into 1s and vice-versa).

Despite considerably common, the use of ROC curves requires additional considerations when dealing with heavily imbalanced classification datasets (where some classes are greatly overrepresented) $(25, 67, 68)$. For instance,consider the case where $NEG \gg POS$. Since the denominator of TNR is far greater than the denominator of TPR, a change in TN (for example, missing one more negative label) will have a much smaller impact on TNR than a change in TP (missing a positive label) will have on TPR. Specifically, an increase of k in TN will cause an increase in TNR $\frac{NEG}{POS}$ times greater than the increase in TPR caused by the same k increase in TP.

Given that ROC equally considers both metrics, classifiers that are more sensitive to positive labels will arguably be favored over those prioritizing negative outputs. By the same logic, ROC will also be more lenient towards false positives (that decrease TNR) rather than false negatives (that decrease TPR) (23).

To address this issue, it is commonly suggested the usage of precision-recall (PR) curves instead $(11,19,23,25,27,28)$, where the precision is plotted as the vertical coordinate while the *recall* (another name for TPR) is represented horizontally. The FP term represents the true negatives for a fixed test set (FP = N - TN). Notice that FP in the definition of precision (Equation 2.36) is not divided by the total number of negative labels, as TN is in TNR. Thus, it is argued that AUPR is less likely than AUROC to prioritize the minority class in imbalanced scenarios (23, 25).

We explore these claims in further detail in the following section, formally defining the ROC and PR curves in terms of ideal label probability distributions.

2.7.3.1 Ideal descriptions of AUROC and AUPRC

Consider a general estimator outputting a decision value $s \in \mathbb{R}$ for each input instance. The final predicted class $\hat{y}$ is still to be defined by a threshold s^* so that $\hat{y} = \mathbb{I}(s > s^*)$. For the ideal case of an infinite number of test samples, the possible scoring results of the estimator would be fully determined by the two theoretical distributions of s given the true label y, i.e. the probability density functions $P(s \mid y = 1)$ and $P(s \mid y = 0)$. Similar to 22, we thus define the probability density functions f_k and their corresponding cumulative distribution functions F_k for each of the two classes $k \in \{0, 1\}$:

$$f_0(s) = P(s \mid y = 0) \tag{2.38}$$

$$f_1(s) = P(s \mid y = 1) \tag{2.39}$$

$$F_0(s) = \int_{-\infty}^{s} f_0(u)\, du \tag{2.40}$$

$$F_1(s) = \int_{-\infty}^{s} f_1(u)\, du \tag{2.41}$$

We further define $p = \frac{P}{T} = P(y_{\text{test}} = 1)$ and $n = \frac{N}{T} = P(y_{\text{test}} = 0)$, the fractions of positive and negative labels in the test set, respectively. We can then express the expected values of the confusion matrix as functions of a given threshold s^* for the decision value s:

$$\text{TP}(s^*) = p(1 - F_1(s^*)) \tag{2.42}$$

$$\text{TN}(s^*) = nF_0(s^*) \tag{2.43}$$

$$\text{FP}(s^*) = n(1 - F_0(s^*)) \tag{2.44}$$

$$\text{FN}(s^*) = pF_1(s^*) \tag{2.45}$$

which, in turn, yields

$$\text{TPR}(s^*) = 1 - F_1(s^*) \tag{2.46}$$

$$\text{TNR}(s^*) = F_0(s^*) \tag{2.47}$$

$$\text{FPR}(s^*) = 1 - F_0(s^*) \tag{2.48}$$

$$\text{FNR}(s^*) = F_1(s^*) \tag{2.49}$$

$$\text{Pr}(s^*) = \frac{p(1 - F_1(s^*))}{p(1 - F_1(s^*)) + n(1 - F_0(s^*))} \tag{2.50}$$

The AUROC can now be expressed as

$$\text{AUROC} = \int_0^1 \text{TPR}(\text{FPR})\ d\text{FPR} = \int_{\infty}^{-\infty} \text{TPR}(s)\frac{d\text{FPR}(s)}{ds}\ ds = \\ = \int_{-\infty}^{\infty} (1 - F_1(s))f_0(s)\ ds \quad (2.51)$$

The integration limits arise from the fact that FPR is maximal when all instances are classified as positives and minimal when all instances are classified as negatives, which respective corresponds to $s^* \to -\infty$ and $s^* \to \infty$. This formulation of AUROC by Equation 2.51 leads to the most common intuition behind the metric. The first factor $(1 - F(s^*))$, represents the fraction of positive instances that are ranked higher than the threshold s^*, alternatively expressed as $\int_{s^*}^{\infty} f_1(s)\ ds$. f_0, as defined by Equation 2.41, represents the probability of finding a negative instance within s^* and $s^* + ds$. Hence, the product of both factors represents the joint probability of having a negative instance between s^* and $s^* + ds$ while also finding a positive instance with $s > s^*$. After integration over all possible thresholds, we conclude that the AUROC score represents the overall probability of randomly selecting a positive instance ranked higher than a randomly selected negative instance.

The baseline score 0.5 can be derived as follows. A random classifier is defined as an estimator incapable of distinguishing between the true label distributions of each class. That is, a classifier is a random classifier if and only if $f_0(s) = f_1(s)\forall s$. As a consequence,

$$\text{AUROC} = \int_{-\infty}^{\infty} (1 - F_0(s))f_0(s)\ ds = -\int_{-\infty}^{\infty} (1 - F_0)\ d(F_0) = \left[\frac{(1 - F_0)^2}{2}\right]_{-\infty}^{\infty} = \frac{1}{2} \quad (2.52)$$

Equation 2.51 also shows a characteristic of AUROC discussed in the previous section: the AUROC score is independent of the relative prevalence of each class in the test set. There is no influence of p or n and only the dependency on the learned decision value distributions.

The AUPR is defined as

$$\text{AUPR} = \int_0^1 \Pr(\text{TPR})\ d\text{TPR} = \int_{\infty}^{-\infty} \Pr(s)\frac{d\text{TPR}(s)}{ds}\ ds = \\ = \int_{-\infty}^{\infty} \Pr(s)f_1(s)\ ds = \int_{-\infty}^{\infty} \frac{p(1 - F_1(s))f_1(s)\ ds}{p(1 - F_1(s)) + n(1 - F_0(s))} \quad (2.53)$$

From Equation 2.53, AUPRC can be interpreted as the average precision weighted by the distribution of positive instances. Analogously, it corresponds to collecting the decision values attributed to each positive instance, and then calculating the average precision considering only these values as classification thresholds. Furthermore, unlike AUROC, the true label cumulative distributions (F_0 and F_1) appear each weighted by their respective class prevalences (p and n).

2.7.3.2 AUPR and AUROC in terms of ranked decision values

When computing AUROC or AUPR for a given estimator on a test set, the values of the decision function s are usually not directly considered. Instead, the decision values outputted for each test instance are used to rank them from lowest to highest, and from these ranked test labels the curves and respective areas are obtained. The specific values of s are thus indifferent to the scoring process, as long as the ranking is preserved. If the percentile rank of each test instance is denoted by $r \in [0,1]$ and each test instance is associated with a decision value, there is a one-to-one monotonic correspondence between r and s in the limit of an infinite number of test samples. Formally, we have

$$r(s) = pF_1(s) + nF_0(s) \tag{2.54}$$

$$dr = [pf_1(s) + nf_0(s)]\, ds \tag{2.55}$$

We also take the liberty to represent $g(r) = g(s(r))$, so that

$$F_1(r) = \int_0^r \frac{f_1(r)}{pf_1(r) + nf_0(r)}\, dr \tag{2.56}$$

$$\frac{dF_1(r)}{dr} = \frac{f_1(r)}{pf_1(r) + nf_0(r)} \tag{2.57}$$

AUPR as defined in Equation 2.53 can now be written as

$$\begin{aligned}
\text{AUPR} &= p\int_0^1 \frac{1 - F_1(r)}{(p+n) - [pF_1(r) + nF_0(r)]}\, \frac{f_1(r)}{[pf_1(s) + nf_0(s)]}\, dr = \\
&= p\int_0^1 \frac{[1 - F_1(r)]}{[1-r]} \frac{dF_1(r)}{dr}\, dr = -\frac{p}{2}\int_0^1 \frac{1}{1-r}\, \frac{d[1-F_1(r)]^2}{dr}\, dr = \\
&= -\frac{p}{2}\left|\frac{[1-F_1(r)]^2}{1-r}\right|_0^1 + \frac{p}{2}\int_0^1 \left[\frac{1-F_1(r)}{1-r}\right]^2 dr = \\
&= \frac{p}{2}\left\{1 + \int_0^1 \left[\frac{1-F_1(r)}{1-r}\right]^2 dr\right\} = \\
&= \frac{p}{2}\left\{1 + \int_0^1 [\Pr(r)]^2 dr\right\}
\end{aligned} \tag{2.58}$$

in which the upper boundary term is determined by using L'Hôpital's rule and noticing that $F_1(r=1) = 1$ and $\frac{dF_1(r)}{dr} \leq \frac{1}{p}$.

$$\lim_{r \to 1} \frac{[1-F_1(r)]^2}{1-r} = 2 \cdot \lim_{r \to 1} [1 - F_1(r)]\, \frac{dF_1(r)}{dr} = 0$$

Equation 2.58 reveals that AUPR is closely related to the average squared precision across all ranks.

We can also express the AUROC in terms of the percentile ranks:

$$\begin{aligned}
\text{AUROC} &= \frac{1}{n}\int_0^1 (1-F_1(r))\left(\frac{nf_0(r)}{nf_0(r)+pf_1(r)}\right)\,dr = \\
&= \frac{p}{n}\int_0^1 (1-F_1(r))\left(1-\frac{pf_1(r)}{nf_0(r)+pf_1(r)}\right)\,dr = \\
&= \frac{1}{n}\int_0^1 (1-F_1(r))\left(1-p\frac{dF_1(r)}{dr}\right)\,dr = \\
&= \frac{1}{n}\int_0^1 (1-F_1(r))\,dr - \frac{p}{n}\int_0^1 (1-F_1(r))\,d(F_1(r)) = \\
&= \frac{1}{n}\int_0^1 (1-F_1(r))\,dr + \frac{p}{n}\left[\frac{(1-F_1(r))^2}{2}\right]_0^1 = \\
&= \frac{1}{n}\left\{\int_0^1 (1-F_1(r))\,dr - \frac{p}{2}\right\} \qquad (2.59)
\end{aligned}$$

Equations 2.58 and 2.59 put AUPR and AUROC in a similar format, better delineating the differences between the two metrics. Consider expressing both now in terms of the precision.

$$\text{AUROC} = \frac{1}{n}\left\{\frac{1}{p}\int_0^1 (1-r)\text{Pr}(r)\,dr - \frac{p}{2}\right\} \qquad (2.60)$$

$$\text{AUPR} = \frac{1}{2}\left\{p + \frac{1}{p}\int_0^1 \text{Pr}(r)^2 dr\right\} \qquad (2.61)$$

Ignoring constant terms and factors, both metrics are centered on integrating the precision over all possible ranks, each rank representing a classification threshold. The crucial difference is that AUPR equally considers all precision values, while AUROC weights each $\text{Pr}(r)$ value by the the reversed ranks.

The precision metric is normalized by the number of instances it considers, so precision values in different r have comparable magnitude. With this in mind, notice that the precision values calculated for $r \approx 1$ are obtained from a very small number of labeled instances. As a result, each label considered when $r \approx 1$ has a large influence on the $Pr(r)$ value, and precisions will tend to have larger variances when $r \approx 1$. Therefore, the AUPR metric is more influenced by the label value of the highest ranked instances, since it equally considers all precision values.

Additionally, the quadratic exponent of AUPR's integrand emphasizes higher precision values overall, independently of the number of samples on which they were calculated $(1-r)$. This would amplify the effect of high $Pr(r)$ values for $r \approx 1$ resulting from stochastic label variations. Hence, we suggest that the AUPR metric is more prone to noise in the labels of higly ranked samples, prioritizing models that strictly maintain high precision for a smaller selection of highest ranks.

On the other side, AUROC compensates this effect by weighting each $\text{Pr}(r)$ value by $1-r$, so that precision values calculated for lower ranks are prioritized. The prioritized precision values ($r \approx 0$) are calculated from a larger number of labels, having lower variances and being more robust to label noise. Each label considered when $r \approx 0$ has a smaller influence on the $Pr(r)$ value relative to the other labels being considered. However, AUROC assigns a larger weight to these precision values, so the influence of smaller ranks also tend to increase.

This characteristic of AUROC might be more suitable for model comparison under the ositive-unlabeled assumption, even though AUPR is usually reccomended for imbalanced scenarios (25, 67, 68). In PU datasets, it is common for negative-labeled instances to rank highly due to the possibility of them being unannotated positives. These highly-ranked negatives would have a larger influence on the AUPR score in comparison to the AUROC score. In fact, we show in the next section that AUROC is closely related to the Mean Percentile Rank metric, which has been suggested for PU learning contexts of interaction prediction and recommendation systems (19, 26–28). Section 2.9.1 presents further discussion on the specific usecases of AUPR and AUROC, besides a numerical analysis of their dependence on the percentile ranks.

2.7.3.3 AUROC is the normalized mean percentile ranks

From Equation 2.59, we can express AUROC as

$$\begin{aligned}
\text{AUROC} &= \frac{1}{n}\left\{\int_0^1 (1 - F_1(r))\, dr - \frac{p}{2}\right\} = \\
&= \frac{1}{n}\left\{1 - \int_0^1 F_1(r)\, dr - \frac{p}{2}\right\} = \frac{1}{n}\left\{1 - \int_0^1 \frac{d(r)}{dr} F_1(r)\, dr - \frac{p}{2}\right\} = \\
&= \frac{1}{n}\left\{1 - [rF_1(r)]_0^1 + \int_0^1 r\frac{dF_1(r)}{dr}\, dr - \frac{p}{2}\right\} = \\
&= \frac{1}{n}\left[\int_0^1 r\, dF_1 - \frac{p}{2}\right] \qquad (2.62)
\end{aligned}$$

Equation 2.62 offers another perspective on AUROC. The term $\int_0^1 r\, dF_1(r)$ represents the expected percentile rank of the positive samples:

$$\int_0^1 r\, dF_1(r) = E[r \mid y = 1] = \text{MPR} \qquad (2.63)$$

This quantity is sometimes referred to as the *mean percentile ranking* (MPR) in the previous literature (26), being proposed in contexts of recommendation systems (26, 69) and bipartite interaction prediction (11, 27, 28).

Consider now the maximum and minimum values of MPR, achieved, respectively, for the ideal $f_1(r)$ distributions:

$$f_{1,\text{max}}(r) = \frac{1}{p}\mathbb{I}(r > n) \qquad f_{0,\text{max}}(r) = \frac{1}{n}\mathbb{I}(r < n) \qquad (2.64)$$

$$f_{1,\text{min}}(r) = \frac{1}{p}\mathbb{I}(r < p) \qquad f_{0,\text{min}}(r) = \frac{1}{n}\mathbb{I}(r > p) \qquad (2.65)$$

Applying these definitions to determine $\frac{dF_1(r)}{dr}$ and using the results in Equation 2.63, we obtain

$$\mathrm{MPR}_{\mathrm{max}} = \frac{1}{p}\int_n^1 r\,dr = \frac{1-n^2}{2p} = \frac{1+n}{2} \tag{2.66}$$

$$\mathrm{MPR}_{\mathrm{min}} = \frac{1}{p}\int_0^p r\,dr = \frac{p}{2} \tag{2.67}$$

from which is straightforward to show that

$$\mathrm{AUROC} = \frac{\mathrm{MPR} - \mathrm{MPR}_{\mathrm{min}}}{\mathrm{MPR}_{\mathrm{max}} - \mathrm{MPR}_{\mathrm{min}}} \tag{2.68}$$

Therefore, the AUROC can also be interpreted as the normalized MPR. To the best of our knowledge, this relationship is not clearly shown in previous explorations.

This result corroborates the argument that the AUROC could be preferable for PU learning scenarios, since the closely-related MPR is a known metric specifically recommended for this context (19, 26, 28, 64). Furthermore, the normalization enables the comparison across different tasks, ensuring that the scores on each dataset are always in the same range. Hence, the AUROC should be preferred over the MPR for the majority of cases.

Related experiments

- **Section 2.9.1 – What are the differences between AUROC and AUPR?**

  Empirically assesses the relative dependence on different ranks of AUPR and AUROC, and discusses the specific usecases of each metric.

2.8 Experimental settings

This section describes the datasets and the evaluation settings used in this study. Experimental results will be presented in Section 2.9.

2.8.1 Datasets

We gathered ten publicly available interaction datasets to evaluate the performance of the proposed models. Quantitative information about each of them is presented by Table 2, and more detailed descriptions are provided in this section.

- **DPI-E, DPI-G, DPI-I, DPI-N (70)**

  These datasets comprise drug-protein interactions for four distinct classes of proteins: enzymes, GPCRs, ion channels, and nuclear receptors, respectively. Drug similarities were computed using the SIMCOMP metric, while protein similarities were computed as normalized scores of Smith-Waterman pairwise alignments (70).

Table 2 – Summary of the datasets used in this study. The similarity scores for mirTarBase and NPInter were obtained from the raw sequences as their normalized Smith-Waterman alignment scores. See Section 2.8.1 for more information.

Dataset	Type of interaction	Y shape	Density	References
DPI-E	Drug-enzyme	664×445	0.9902%	(70)
DPI-G	Drug-GPCR	95×223	2.997%	(70)
DPI-G	Drug-GPCR	95×223	2.997%	(70)
DPI-I	Drug-ion channel	204×210	3.445%	(70)
DPI-N	Drug-nuclear receptor	26×54	6.410%	(70)
ERN	Gene-transcription factor	1164×154	1.837%	(8)
SRN	Gene-transcription factor	1821×113	1.780%	(17, 71–74)
DAVIS	Inhibitor-kinase	68×442	5.011%	(61, 75, 76)
KIBA	Inhibitor-kinase	2111×229	19.74%	(61, 76, 77)
NPInter	lncRNA-protein	586×446	18.12%	(78, 79)
mirTarBase	miRNA-mRNA	1873×415	7.065%	(80, 81)

Source: By the author.

- **ERN (8) and SRN (17, 71–74)**

  The datasets represent interactions between genes and transcription factors in *E. coli* and *S. cerevisiae*, respectively. Features for genes and trasncription factors are initially composed of experimentally measured expression levels and, in SRN, gene motif features (17, 82). We compute the RBF kernel of such values to obtain the final similarity matrices.

- **DAVIS (19, 75)**

  The DAVIS dataset contains experimentally measured drug-kinase dissociation constants (75). The dataset was binarized by considering interactions with dissociation constants $\leq 30nM$ as the positive ones, as suggested by (19). Drug similarities were computed using the Extended Connectivity Fingerprints (ECFP4) (19, 83) while protein similarities were taken as the normalized Smith-Waterman score (19, 70).

- **KIBA (61, 76, 77)**

  The KIBA dataset was initially built by 77 and contains experimentally verified affinity scores between kinase and kinase inhibitors.

  61 further processed the dataset by removing all drugs and targets with less than 10 observations. In alignment with 61, 77, we consider positive interactions as those with $\log_{10}$ KIBA-scores ≤ 3.0 to reframe the task as binary classification.

  The utilized version of the dataset with corresponding amino acid sequences and SMILES representations were provided by 76. From them, we generated the protein similarity matrix using the same procedure employed in the preprocessing of NPInter proteins. The drug similarities were computed similarly to how 19 processed the DAVIS dataset, using the Tanimoto distances of ECFP4 fingerprints (19, 83). The Python library `rdkit` (84) was used to this calculation.

- **mirTarBase (80, 81)**

  The mirTarBase dataset contains experimentally validated microRNA-messengerRNA interactions. MicroRNA sequences were obtained from miRBase (85) while transcript sequences were obtained from GENCODE (86). The longest transcript for each gene was selected and the 3' UTR exonic sequences were recovered from the genome and annotation files provided by GENCODE. The similarity matrices were then built from the normalized Smith-Waterman (70) alignment scores among microRNAs and among the genes' 3' UTRs. The alignments were performed using the BLASTN substitution matrix and no gap penalty, with the help of the Biopython package (87).

  Each miRNA was required to have at least 10 interactions in the dataset, and each gene was required to have at least 100 interactions.

- **NPInter (78, 79)**

  Interactions between long non-coding RNAs (lncRNA) and proteins were recovered from NPInter (78,79). The lncRNA sequences were obtained from NONCODE (88,89) and the protein sequences were obtained from UniProt (90). The similarity matrices were built from the normalized Smith-Waterman (70) alignment scores among lncRNAs and among the proteins. Similarly to the preprocessing of mirTarBase, we utilized the Biopython package (87) to perform the alignments. using the BLASTN and BLOSUM62 substitution matrices for the lncRNA and protein alignments, respectively, and no gap penalty in both cases.

  Each lncRNA was required to interact with 50 proteins or more to be incorporated in the dataset, and each protein was required to have at least 2 interactions.

2.8.2 Evaluation procedure

Unless otherwise specified, the comparison experiments in the following sections were performed using a 4 by 4 cross-validation scheme (Section 2.7.2). The 10 datasets described in Section 2.8.1 had their rows and columns randomly permuted before being partitioned into the 16 bidimensional folds. The same permutation was used for all the estimators.

Each fold was selected once as the TT set, yielding 16 partitionings of the data into TT, LT, TL and LL sets (see Section 2.7.1). The models were trained on each LL partition and evaluated on the three remaining ones. Results for LT and TL were averaged to produce the LT+TL scores (Section 2.7.1).

The metrics AUROC and AUPR were utilized for evaluating predictions, as defined in Section 2.7.3. Each model then produces 16 scores for each dataset, totalling 160 values per model for each of the four test setting: TT AUROC, TT AUPR, LT+TL AUROC and LT+TL AUPR.

From the metrics definitions (Equation 2.59 and Equation 2.58), we see that they depend on the class prevalences of the test set, therefore being not directly comparable across datasets. To obtain comparable scores, we use percentile ranks for each fold, and employ rank-based test statistics.

Specifically, a scenario with m models. For each fold in each dataset, we sort the m scores and substitute their original value with the percentile ranks. Now for each estimator and each dataset, the percentile ranks of the 16 folds are averaged, resulting in a single rank for each estimator and dataset. Each estimator then has 10 average ranks, one value for each dataset. Each average rank represents the number of victories against the other estimators. These distributions of 10 values per estimator are what we utilize for the statistical tests, so that the number of folds does not influence the significance estimates.

We obtain omnibus p-values through a Friedman test (91, 92), followed by pairwise Wilcoxon rank-sum tests (92) with Benjamini-Hochberg correction (4, 5).

In Sections 2.9.6, 2.9.7 and 2.9.8, we also investigate the impact of missing positive annotations. To do so we randomly mask a fraction of the positive labels from the training set before training the model on each fold. All models receive the same mask for each fold, and only the LL set is altered (not the test sets used for evaluation). Four settings are explored: we select 0%, 50%, 70% and 90% of the positive annotations to randomly turn into zeros. Folowing 93, we call the dropout fractions the *incomplete label ratio* (IRL), and include the percentage at the end of the metric name to indicate the setting (for example, TT AUPR 50% or LT+TL AUROC 70%).

2.9 Experiments

This section describes the experiments performed in this work, designed to explore the proposed research questions (Section 1.6), assess the effectiveness of the developed methods, and to validate predictions made in the theoretical analyses.

2.9.1 What are the differences between AUROC and AUPR?

This experiment was designed to evaluate the hypotheses raised in Section 2.7.3.2, comparing the AUROC and AUPR metrics in terms of their dependence on the percentile ranks of the predictions. We aim to elucidate the differences between the two metrics and to provide insights on when to use each of them.

Key findings

- AUPR prioritizes a smaller number of highest-ranked interactions, while AUROC considers a larger number of both highest and lowest ranks.
- AUPR should be used when the goal is to select a small number of most-likely interactions. AUROC should be used i) when both likely-positive and likely-negative interactions are important; or ii) when interested in a large fraction of the predictions.
- AUROC could be also preferable for PU learning.

2.9.1.1 Setup

Our main objective is to elucidate how much importance each metric assigns to each percentile rank. To do so, we perform a Monte Carlo simulation. First, we select R equally spaced rank values between 0 and 1 (excluding 0 and 1). For each rank, we generate N random binary values to be used as labels. Therefore, each of the N iterations of the simulation will produce a random set of R binary labels, corresponding to each percentile rank. In each iteration, we use the R labels and R percentile ranks to calculate AUROC and AUPR, resulting in N values for each metric. Finally, for each of the R ranks, we calculate the point biserial correlation (94) between the N random labels and the N AUROC and AUPR values.

2.9.1.2 Discussion

The results are displayed in Figure 8. The plot shows that AUPR is highly sensitive to a small set of highest ranks, while ranks closer to 0 have little influence on the metric. On the other hand, the correlations with the AUROC score are more distributed across ranks. Also, for AUROC, both higher and lower ranks have a higher impact, but in opposite directions. This behaviour in agreement with the theoretical analyses from Section 2.7.3.2. The correlation with AUROC is also symmetrical around the median ($r = 0.5$), linearly increasing from arround -0.05 at $r = 0$ to around 0.05 in $r = 1$. This demonstrates the class symmetry of the AUROC metric described in Section 2.7.3.2, meaning that negative and positive labels could be swapped without affecting the final results.

Overall, these results demonstrate that using AUPR is more suited when one is interested in selecting a restricted number of top-ranked instances from a pool of predictions, such as in recommendation systems or drug discovery tasks. Conversely, AUROC should be favored when the goal is to rank a large batch of interactions. Examples would be modeling genetic interactions in a genome-wide fashion or building interaction databases.

If one intends to select a small number of negative instances instead, a possible strategy would be to swap the binary labels and use AUPR. However, one should prioritize AUROC over AUPR if the goal is to select both negative and positive predictions with the highest confidences.

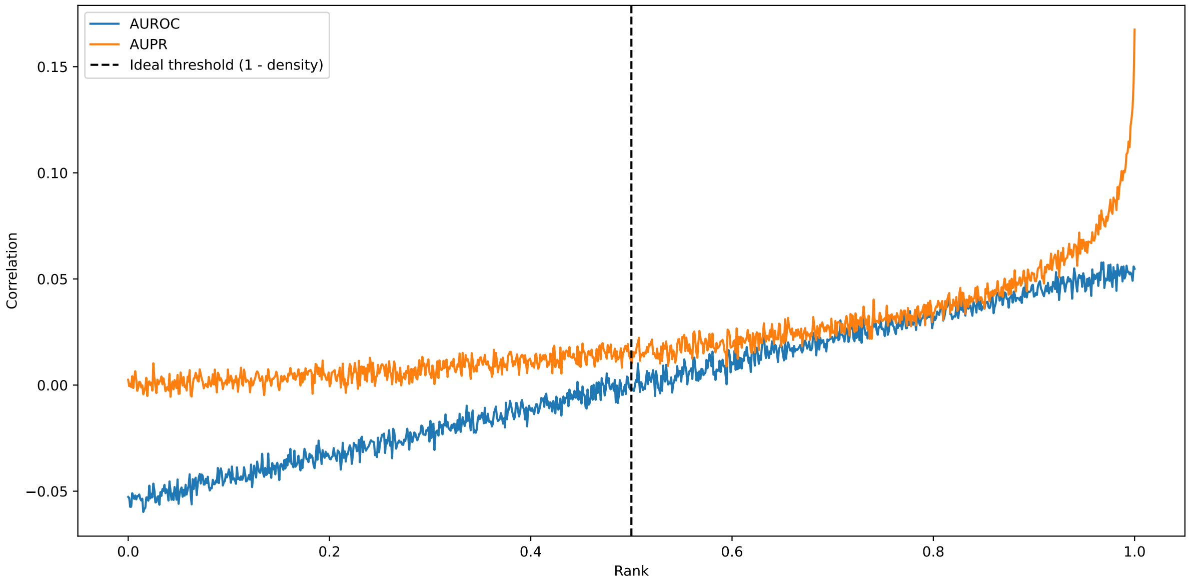

Figure 8 – Point biserial correlation between the binary labels at each percentile rank and the AUROC and AUPR scores. The correlation was calculated between the ranks and the scores for $N = 10^5$ random sets of $R = 10^3$ binary labels. The results show that AUPR is very sensitive to a small group of highest ranks, while AUROC has a more distributed weigthing profile. For AUROC, both higher and lower ranks have a higher impact than ranks around 0.5, contributing in opposite directions to the final metric. The results confirm the theoretical analyses from Section 2.7.3.2.
Source: By the author.

This conclusion results from AUPR disregarding ranks close to 0, so models selected with AUPR are not guaranteed to yield the most confident true negatives.

We also argue that AUROC could be preferable for comparing models under the PU assumption, at least in purely theoretical settings. For PU datasets, we naturally expect some negative-labeled instances to be very highly ranked since they could be, in fact, unannotated positives. AUPR would more strictly penalize such predictions, favoring models that consider the labeling mechanism itself rather than only the underlying interaction mechanism. AUPR could thus undermine the model's potential to discover new interactions, possibly failing to gauge the generalization capabilities of algorithms in a PU context. The results from Section 2.9.8 seem to corroborate this hypothesis, with AUROC ranking estimators more consistently than AUPR across different levels of label noise.

2.9.2 Are BGSO models faster than GMO models?

This experiment empirically measures the training time complexity of the tree models under study. Our main goal is to validate the theoretical complexity analysis presented in Section 2.4.2 and to compare the training times of the proposed BGSO models with the GMO models.

Key findings

- The GMO complexity is measured to be $O(n^3 \log n)$, while the BGSO complexity is measured to be $O(n^3)$.
- The empirical results match the theoretical expectations from Section 2.4.2.
- The difference in complexity between GMO and BGSO is less pronounced for the Extra-Trees, but the BGSO models still present significantly lower complexity than the GMO models.

2.9.2.1 Setup

We artificially generate a series of bipartite datasets by filling three n by n matrices with pseudo-random values, representing the two X matrices and the Y matrix on each interaction. Values were taken uniformly from the interval $[0, 1]$ for the feature matrices and from the interval $[0, 100]$ for the target matrix. We then train the GMO and the BGSO versions of a single bipartite decision tree (BDT) and a single bipartite Extra-Tree (BXT) on each of the generated datasets, measuring their training duration in seconds.

2.9.2.2 Discussion

The results are shown in Figure 9. From the least squares linear regression on the log-log plot, we see that the estimated training time complexities closely follow the theoretical expectations developed under Section 2.4.2, with slopes referring to the BGSO models (predicted to be $O(n^3)$) approaching 3 while the GMO models (predicted to be $O(n^3 \log(n))$) produce slope between 3 and 4.

Statistical testing further shows that the empirical time complexity of the proposed BGSO algorithms are indeed significantly lower than that of their GMO counterparts (Figure 9). We also notice slightly lower slopes for the Extra-Trees in comparison with the BDTs. In fact, they seem to be closer to the theoretical expectations for the BGSO models. This is expected, since the randomized split search performs far less operations than the exhaustive search (Section 2.3.1). Notwithstanding, both algorithms have the same order of complexity. We then suggest that much larger datasets would be required to observe the asymptotic behavior of the Extra-Trees. In spite of that, the empirical complexity of `bdt-bgso` is still observed to be highly significantly lower than that of `bxt-sq`, validating once more the prediction that `bdt-bgso` should present faster training times than `bxt-sq` on sufficiently large datasets.

2.9.3 Which prototype should a GMO forest use?

In this work we propose a different prototyping strategy to determine the output value of each leaf in a GMO decision tree, taking the similarity matrices of our use cases into consid-

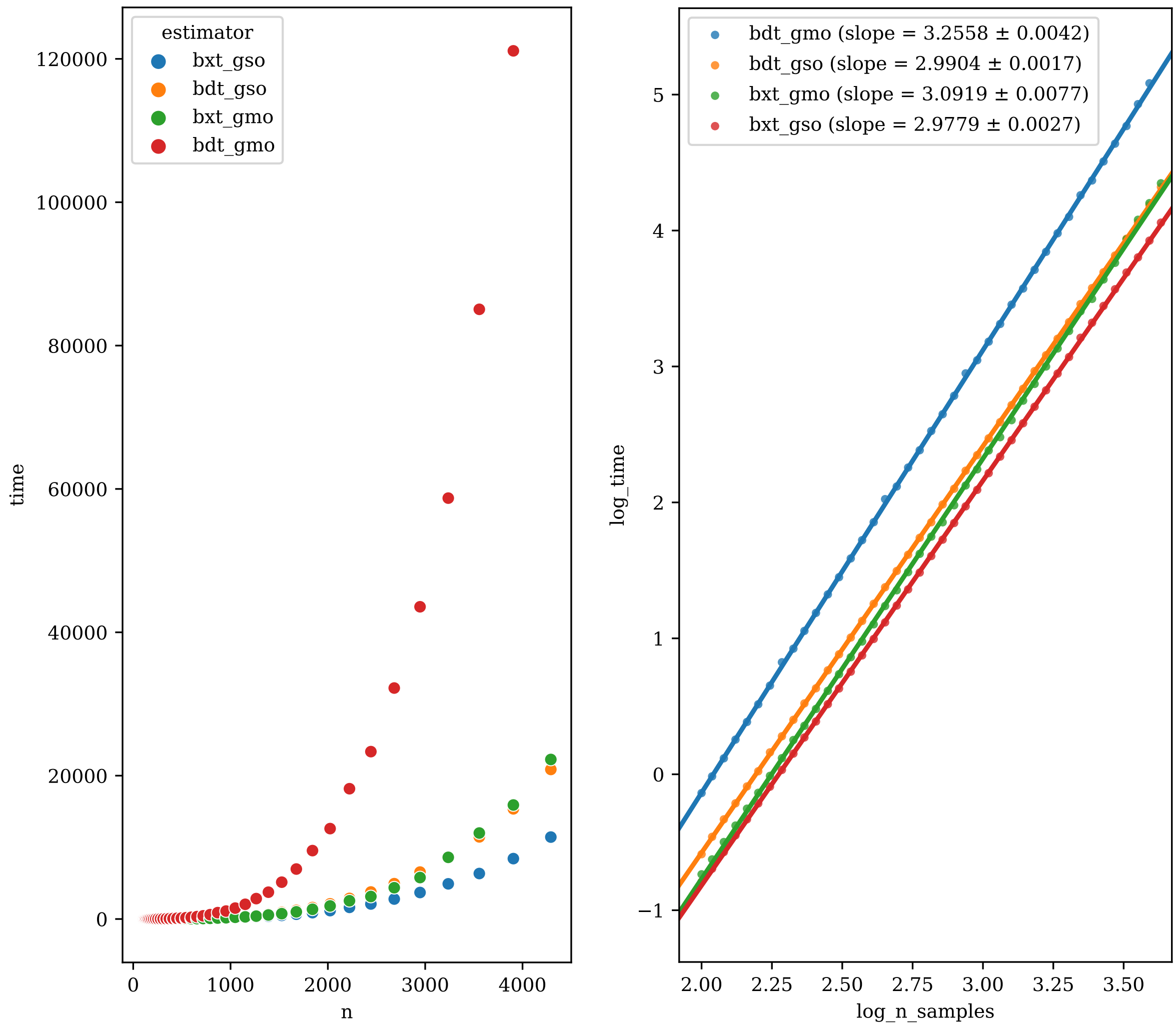

Figure 9 – Empirical time complexity estimation of the proposed bipartite global single-output (BGSO) and the global multi-output (GMO) (1) algorithms. Bipartite versions of both extremely randomized trees (2) (BXT) and greedy decision trees (3) (BDT) were built under the BGSO and GMO scheme and trained over artificial datasets of varying numbers of samples (as described in Section 2.4.2). The training time versus the number of samples is presented on the left. On the right, we present the same plot on a logarithmic scale. The slopes and respective standard deviations are obtained by applying least-squares linear regression. Independent two-sample t-tests comparing the slope estimates reveal that the time complexity of `bdt-bgso` is highly significantly lower than `bdt-gmo` ($p < 10^{-64}$) and even `bxt-gmo` ($p < 10^{-20}$), and also that `bxt-gso` significantly exhibits lower complexity than `bxt-gmo` ($p < 10^{-22}$).
Source: By the author.

eration (Section 2.4.3). In this experiment we compare their predictive performances, building BXT and BRF models for every option.

Key findings

- The `square` strategy is the best for both BXT and BRF models, except for the LT+TL AUPR score, for which the fully-grown trees are the best.

- The weighted-neighbors strategies seem to improve generalization.

2.9.3.1 Setup

The minimum rows per leaf and minimum columns per leaf were both set to 5, ensuring that at least 5 samples of each domain are considered when calculating the prototype values. To observe the effect of this early-stopping criterion by itself, we also include forests of fully-grown trees in the comparison. The compared models are described below.

- **`GMOsa`:** proposed by 1, the output of each leaf is the average of the labels of the learned samples that reach that leaf (Equation 2.16). Their proposal had not been extended yet to ensembles of bipartite trees or to BXT.
- **`uniform`:** also proposed by 1, but in the context of a single tree. It works as `GMOsa` for TT, but for TL and LT the average is taken only among the labels of the known instance (Equation 2.18).
- **`precomputed`:** the labels in each leaf are weighted by the similarities between the learned samples and the test instances (Equation 2.19).
- **`square`:** similar to `precomputed`, but the squared similarities are used instead (Equation 2.19).
- **`softmax`:** also a weighting approach, but using the exponential of the similarities (Equation 2.19).
- **`full`:** the trees are grown until a single interaction remains in each leaf.

2.9.3.2 Discussion

The results for BRF (Figure 10) and BXT (Figure 11) were similar. In all cases except LT+TL AUPR, using the square of the similarities to weight the labels in each leaf resulted in in the best scores. For the LT+TL AUPR score, growing the decision tree to its maximal size was the best strategy, followed by `uniform`, the original proposal by 1 of averaging only the outputs of the learned samples (known from the training set) in each leaf. With the exception of the LT+TL AUROC metric for BXT, the mentioned winning models were statistically distinguished from all the remaining estimators. For LT+TL AUROC, the superiority of `square` could not be attested when compared to the `uniform` strategy.

The fully-grown versions of both forest algorithms are shown to be especially advantageous when considering the LT+TL AUPR. This suggests that building trees to their maximum depth is the best strategy for learning tasks in which

1. one of the domains is fixed, with the final goal being to model how new instances will bind to this known set of entities;

2. the goal is to select a small number of top-ranked interactions (see Section 2.9.1).

On the other hand, weighted averages seem to improve the forest's generalization ability, as they seem to perform best under the AUROC metric and TT contexts.

We propose that this distinction results from the ability of a fully-grown tree to independently consider the labels of each learned instance when calculating the prototype. However, the methods using weighted averages ivariably mix the labels of a pool of neighbors in each leaf. This hypothesis is supported by the fact that the second-best model regarding LT+TL AUPR is the `uniform` strategy, which also uses the labels of individual learned samples to generate the predictions. The hypothesis alone, however, do not explain the superiority of the deeper trees over the `uniform` weights. In this case, the larger tree depth is likely beneficial through i) an increase in the predictive power of each individual tree, and/or ii) an increase in tree diversity, both of which would improve the ensemble's performance as discussed in Section 2.5. We let to future work the more specific investigation of these effects.

On the other hand, `full` and `uniform` show notable inferior performance in the TT evaluation settings, suggesting that these strategies have inferior ability to consider completely new interacting pairs.

We select the squared weighting strategy and the fully grown trees to be further investigated in the downstream analyses.

2.9.4 Which adaptation strategy is the best for decision forests?

We now compare each of the described approaches for adapting decision forests to bipartite data, including data-centric adaptations (Section 2.2) and the natively bipartite forests (Section 2.4 and Section 2.4.1).

Key findings

- SLMO is the best strategy for BRF models on LT+TL sets.
- GMO is the best strategy for completely new dyads.
- Undersampling of negative annotations is beneficial for a GSO BXT in terms of AUROC, but should be avoided if the goal is to select the highest ranked interactions.

2.9.4.1 Setup

We briefly describe below the suffixes in the model names of this section, indicating the employed bipartite adaptations.

- **`slmo:`** implements the standard local multi-output approach (SLMO; Section 2.2.2) by training four separate multioutput models, two for each domain. First explored by 17.
- **`slso:`** also implements the SLMO approach (Section 2.2.2), but each multioutput model is instead a composition of several local single-output (SLSO) models, i.e. one model is trained for each row or column of the interaction matrix. This setting is similar to the early proposal by 95, but employing decision forests as the base algorithm.
- **`gmo:`** a fully-grown global multi-output forest, as explored by 18 under the name eBICT.
- **`sq:`** a global multi-output forest with minimum leaf dimensions of 5 by 5, implementing our proposed squared-similarities weighting for the prototype function (Section 2.4.3).
- **`bgso:`** a bipartite global single-output forest, implementing our proposed algorithm with improved computational complexity over SGSO and GMO (Section 2.4.1).
- **`sgso-us:`** implements the standard global single-output adaptation (17), additionally employing undersampling of the negative annotations to yield a balanced training set (see Section 2.2.1).

All the global models were built with 100 trees. For SLMO, each of the four forests used 50 trees, while for SLSO 50 trees were used for each row or column of the interaction matrix.

2.9.4.2 Discussion

In the LT+TL AUPR evaluation setting, the pattern observed in Section 2.9.3 again emerges: the label averaging strategy employed by SQ performs considerably worse in comparison to forests that separately consider each known instance. In particular, the SLMO and SLSO adaptations yield the clear best BRF models in terms of LT+TL AUPR, and these adaptations interpret each instance as a separate output to be predicted. Among the local adaptations, SLMO significantly ouperforms SLSO. SLSO treats each training instance as a completely independent task, building a separate forest for each row and column of the interaction matrix. As such, the previous result shows that this complete independence is not desirable for the learning problems under study, and exploring label correlations between instances of the same domain is beneficial (as demonstrated by SLMO). This result could partially be a consequence of the very sparse nature of our problems: if the interaction information of each instance is limited, it becomes advantageus to aggregate information from other instances with correlated interactomes. We then speculate that the advantage of SLMO over SLSO could become less prominent once the number of known interactions per instance increases and more data are available for training.

Still considering LT+TL AUPR but focusing on BXT models instead, we notice that SLMO loses the advantage to the fully-grown bipartite trees SQ and BGSO. A possible explanation comes from the fact that local approaches yield shallower trees, and much more extra-trees could be required to achieve comparable performance. In more detail, first notice that the individual performance of each tree in a BXT forest is lower than that of a tree in a BRF. Thus, a BXT ensemble requires a larger number of trees to reach satisfactory performance. Additionally, if the trees are shallow, they tend to be less representative of the training data (each tree node brings a little more information on the dataset). Even more trees then should be required to compensate the randomness of each individual. Finally, building a forest locally as in SLMO or SLSO results that each tree is trained on a much smaller number of samples in comparison to considering each dyad as a separate instance. Therefore, local approaches generate smaller and less representative trees. We then suggest that the advantage of SLMO could also manifest for BXT estimators if the number of trees in the ensemble were to be increased.

For both BXT and BRF in all the TT settings, the SQ model significantly outperforms the other estimators. SQ is also the best model in the LT+TL AUROC setting when comparing BXTs and the second best for BRFs. This reinforces the findings of Section 2.9.3 suggesting that larger leaves and label weighting seems to improve generalization to unseen instances. We also highlight SLMO as a prominent strategy for BRFs under TT AUROC and TT AUPR, being the second best model in both cases.

The SGSO US strategy performed consistently worse than the others under the AUPR metric. Conversely, it was significantly the second best BXT model in both LT+TL and TT AUROC settings, while surpassing BRF gmo also under both AUROC test sets. This shows that unsersampling is a viable technique when when considering the overall ranking of interactions. However, it is detrimental when the goal is to select the most likely interactions. That is, SGSO US allows a larger number of false positives in the highest ranked positions.

This behaviour is expected. Notice that SGSO US models are trained in a balanced dataset, while the other models are trained in the original dataset where negative annotations are much more frequent. With more training examples, the other models tend to be considerably better in correctly classifying negative annotations. This suggests that AUPR tends to favor models of high specificity, that are strictly avoid predicting false positives.

2.9.5 Can label imputation assist bipartite forests?

In this section, we compare bipartite forests with and without label imputation by matrix factorization, to investigate its impact as a countermeasure to the high label sparsity in the datasets.

Key findings

- Imputing positive annotations with NRLMF improves the performance of bipartite forests in the majority of cases.
- For LT+TL AUPR, the best BRF is `brf-slmo` and the best BXT is `bxt-gmo-yr`.
- SQ-NRLMF is the overall best model for TT.

2.9.5.1 Setup

We employ the adaptations SLMO, BGSO, GMO and SQ (which is GMO our squared similarities prototype). See Section 2.9.4 for a more detailed description of each model.

It was previously suggested that creating a dense representation of the interaction matrix improves the performance of BXT in drug-target interaction tasks (29). To test this result on our datasets, we compare the bipartite forests scores with and without the interaction matrix reconstruction step. As (29), the reconstruction step was performed using neighborhood-regularized logistic matrix factorization (NRLMF) (31).

We performed a randomized search to select hyperparameters for the NRLMF algorithm. 100 different combinations of hyperparameters were evaluated in terms of their resulting mean squared error in a nested bipartite 5-fold diagnonal cross-validation. The best combination of parameters according to the inner CV loop was then used to reconstruct the interaction matrix of each outer CV fold. Only then the resulting matrices were used as the training data for the bipartite forests. Note that a single forest was built per outer CV fold, so that the NRLMF hyperparameter search was performed independently from the downstream forest performance. The hyperparameters λ_1, λ_2, β_1, β_2, and η were all independently sampled from a log-uniform distribution bounded by $\frac{1}{4}$ and 2 (Function TrainNRLMF). The number of latent vector components was set to be equal for both axes, and chosen between 50 and 100. The number of neighbors was randomly selected as 3, 5 or 10 in each iteration, and the maximum number of optimization steps was always set to 100. The parameter α (Function TrainNRLMF), as in the original paper (31), was set to 5. This parameter specifies the emphasis on positive annotations, so each positive annotation contributes 5 times more to the loss function in our settings, as if 5 copies of each positive interaction were present in the training set.

The results for BRF and BXT are shown by figures 14 and 15, respectively. The `yr` suffix to a model indicates that it was trained after Y reconstruction, that is, on the output of NRLMF.

2.9.5.2 Discussion

The interaction matrix reconstruction step is shown to be especially beneficial in terms of AUROC scores, improving this score for almost all forest algorithms investigated. The only exceptions were `bxt-sq` and `brf-sq` in LT+TL ROC AUC, where the improvement is still observed but not statistically significant.

Even without assistance from NRLMF, the `bxt-sq` and `brf-sq` models employing our squared similarities output weighting (Section 2.4.3 and Section 2.9.3) are placed second in their corresponding TT AUROC results, both significantly outperforming all NRLMF-combined models except for their own versions, `bxt-sq-yr` and `brf-sq-yr`. `brf-sq` was also the top performing BRF model in terms of TT AUPR, significantly surpassing all but `brf-sq-yr` and `brf-slmo-yr`. Similarly, `bxt-sq` was the top performing BXT model under the same metric, significantly surpassing all estimators but `bxt-sq-yr`, `bxt-slmo-yr` and `bxt-gmo-yr`.

Regarding TT AUPR, NRLMF still provides a significant improvement for all models except `brf-sq`, `bxt-sq` and `bxt-slmo`. While both `brf-sq` and `bxt-sq` resulted in the highest average ranks for TT AUPR, the top position could not be statistically resolved between the BRFs using SQ, SQ-NRLMF, and SLMO-NRLMF, nor between the BXTs using SQ, SQ-NRLMF, SLMO-NRLMF, and GMO-NRLMF.

In the LT+TL AUPR setting for BXT ensembles, NRLMF is shown to significantly degrade the performance of `bxt-sq`. Therefore, the combination of the squared neighbors prototype and NRLMF likely results in more high-rank false positives under LT+TL, especially since such degradation is not observed under LT+TL AUROC.

For BRF models under LT+TL AUPR, the advantage of employing NRLMF is not clear. BGSO was the only adaptation to significantly benefit from NRLMF in this setting, while sq and SLMO were significantly impaired by it. In fact, the `brf-slmo` model without label matrix reconstruction significantly outperformed all the other random forests for this evaluation setting.

In summary, interaction matrix reconstruction by NRLMF seems to consistently improve BRF and BXT results in terms of AUROC and also the results of BXT regarding LT+TL AUPR. On TT AUPR, while less evidence is found, the results still seem to point in the same direction of a beneficial NRLMF transformation step. On the other hand, the comparisons of BRF under LT+TL AUPR evaluation indicate the opposite conclusion, disfavoring usage of NRLMF especially for `brf-slmo` and `brf-sq`. We thus discourage the application of NRLMF with BRF in scenarios where i) one of the entities of the dyad is always known to the model, such as in drug repositioning; and ii) the main interest is to select a small set of most-likely interactions.

2.9.6 What is the best way of building semi-supervised forests?

In this experiment, we compare the performances of semi-supervised bipartite forests, as presented by Section 2.6.

Key findings

- The MD unsupervised impurity is the best option for LT+TL.
- AD is the most promising strategy for TT.
- The best strategy for determining the supervision balance seems to be either size or fixed, but further investigation is needed.

2.9.6.1 Setup

All forests in this section are based on the BXT BGSO, as presented by Section 2.4 and used in Section 2.9.4. We evaluate all combinations of the three strategies for calculating the unsupervised impurity and the four strategies for determining the supervision balance at each node.

Strategies for determining the unsupervised impurity:

- **`mse`:** it refers to forests using the mean squared error as the unsupervised impurity (Equation 2.21).
- **`md`:** corresponds forests employing the mean distance unsupervised impurity (Equation 2.23).
- **`ad`:** also uses the mean squared error (Equation 2.21), however, the semi-supervised impurity is only calulated twice on each tree node. The search for the best feature and split point in each instance domain uses the supervised impurity only. Then, the semi-supervised impurity is only used to evaluate the best split in each domain. Finally, the two resulting values for the semi-supervised impurity are used to select between the horizontal and vertical split.

Strategies for determining the supervision balance (σ):

- **`fixed`:** the supervision balance (σ) is fixed at 0.5 for mse and md and at 0 for ad.
- **`density`:** σ is determined by the density of positive annotations in each tree node (Equation 2.24).

- **`size`:** σ is determined by the total number of interactions current in the tree node (Equation 2.25).
- **`random`:** σ is set to a random value between 0 and 1 drawn at each tree node (Equation 2.26).

2.9.6.2 Discussion

The comparison results are displayed by Figure 16. Regarding LT+TL AUROC, the 0% ILR and 50% ILR do not result in statistically significant differences in performance. For ILR = 70%, md random and md size significantly outperform the remaining models, while under IRL=90%, md random, md size, and md density are shown to surpass the others. These results indicate that the MD unsupervised impurity is the most suited to the LT+TL AUROC evaluation setting. Furthermore, we see that the random strategy to select the supervision balance is among the best in both ILR = 70% and ILR = 90%, suggesting that increasing diversity among the trees could be the main mechanism behind the improvement of the semi-supervised models, rather than necessarily guessing the best σ at each tree node. Since md size is also among the best, it is not clear if the advantage of md size over the other models is due to a better choice of σ or to a more stochastic nature of the σs it selects. As a future investigation, we suggest using completely random values for the unsupervised impurity, to assess the possibility of tree-diversity being the main factor behind the observed improvements.

As for TT AUROC, only ILR = 90% yielded significant comparisons. In this case, the mse unsupervised impurity with size, random, and density σ selection were the best strategies, surpassing the remaining. MSE thus seems the better option in terms of AUROC for scenarios with very scarse information and completely unknown instances.

Under LT+TL AUPR, md fixed significantly outperformed the other models for ILR = 50% and ILR = 70%. For ILR = 0%, md fixed is also the first place, but could not be statistically resolved from the second place ad size. AD size was also the second best model for ILR = 0%, 50%, 70%, statistically outperforming all but md fixed in ILR = 70%.

With respect to TT AUPR, the AD strategies prevail, being the best four models for ILR = 50% and ILR = 70%, and being among the five best models for ILR = 0%. The presence of AD random among the best models again suggests that tree-diversity is an important factor for performance improvement. It seems that AD models perform overall better according to AUPR in comparison to AUROC. This is consistent with the discussion in Section 2.9.1: AUPR tends to prioritize models more tighly related to the observed labels, assuming less risk of false positives. AD is the strategy that is less influenced by the unsupervised impurity function.

In conclusion, the best unsupervised impurity under LT+TL seems to be the mean distance, and AD seems to be the best for AUPR TT. The best strategy for determining σ seems to be either size or fixed, but more experiments are needed to confirm this finding.

2.9.7 Which strategies are the most promising?

In this experiment, we compare the performances of various forests from the previous sections, including BXT and BRF, forests using semi-supervised impurities, and forests using label imputation by NRLMF.

Key findings

- Semi-supervised impurities are beneficial in terms of AUROC, especially when more annotations are missing.
- Label imputation with NRLMF is better than semi-supervised impurities.
- BXT are superior to BRF in all settings.

2.9.7.1 Setup

For more information on each model see the corresponding section in the list bellow.

- **`bxt-bgso`, `brf-slmo`:** Bipartite forests without label imputation. See Section 2.9.4 for more information.
- **`md-size`, `md-fixed`, `ad-size`, `ad-fixed`:** Bipartite forests employing semi-supervised impurities. See Section 2.9.6.
- **Models with the `yr` suffix:** Bipartite forests using NRLMF to impute positive annotations. See Section 2.9.5.

2.9.7.2 Discussion

The comparisons are presented by Figure 17. The results reveal a clear superiority of forests employing the NRLMF as a label imputation strategy, in comparison to those using semi-supervised impurities. The four models employing NRLMF were the four highest ranked estimators in almost all evaluation settings, the two exceptions being LT+TL AUPR with ILR = 0% and ILR = 50%. In the first exception (LT+TL AUPR 0%), `brf-sq-yr` and `bxt-sq-yr` are the two worst-performing models, whereas `bxt-gmo-yr` and `bxt-bgso-yr` are the first and second best, respectively. In the second exception (LT+TL AUPR 50%), the two best models are the same, and `brf-sq-yr` is still one the worst performers. However, `bxt-sq-yr` jumps to the third best position.

Furthermore, under the other metrics (LT+TL AUROC, TT AUROC, and TT AUPR), `bxt-sq-yr` was the highest ranked estimator in almost all cases, the only exception being TT AUPR IRL=0%, where it was only behind `bxt-gmo-yr`. Notwithstanding, the comparison between `bxt-sq-yr` and `bxt-gmo-yr` was still not statistically significant in this setting. Similarly, no statistical difference is found between these models in the LT+TL AUROC 0% setting, where they also occupy the first positions. The leadership of `bxt-sq-yr` is also not significant in comparison to `bxt-bgso-yr` under TT AUPR 50% and TT AUPR 70%. For all remaining cases where `bxt-sq-yr` was the best model, it was statistically significantly better than all other estimators (TT AUPR 90%, LT+TL AUROC 50%, LT+TL AUROC 70%, LT+TL AUROC 90%, and all TT AUROC).

Another observation is that some semi-supervised impurities significantly improve predictive performance relative to the original BXT BGSO, especially when more annotations are missing. This conclusion is based on the fact that, in all settings with ILR $\neq 0$ except LT+TL AUPR, the original BXT BGSO is among the three worst performers. For all AUROC settings with ILR $\neq 0$, it was the lowest ranked model. On the other hand, md fixed is noted to significantly surpass BXT BGSO in 10 of the 16 evaluation settings, the exceptions being LT+TL AUROC 0%, TT AUROC 0%, TT AUROC 90%, TT AUPR 0%, TT AUPR 70% and TT AUPR 90%. Therefore, although not as effective as the NRLMF reconstruction technique, using semi-supervised trees seems indeed beneficial when label information is scarce.

When comapring the BXT against the BRF models, we notice that `bxt-sq-yr` significantly outperformed `brf-sq-yr` in all settings. The other random forest, `brf-slmo`, was also significantly outperformed by `bxt-sq-yr` in all cases but LT+TL AUPR 0%, where the opposite was observed. Similar results hold for the other BXT models as well: they significantly surpassed `brf-slmo` in the vast majority of test configurations, with the only exception being TT AUROC 0%, in which the comparison between `bxt-gmo-yr` and `brf-slmo` was not significant. These results suggest that BXT models could offer significant advantages over BRF models in the context of interaction prediction. This conclusion is especially relevant given that the BXT training algorithm is considerably faster than the procedure for building BRFs, as discussed in Section 2.4.2.

2.9.8 Can bipartite forests compete with other proposals?

In this section we compare the two most promising methods we developed (`bxt-sq-yr` and `bxt-bgso-yr`) with several prominent models from the literature.

Key findings

- Forests with NRLMF surpass NRLMF alone.
- `bxt-sq-yr` is the best overall performer, with the exception of LT+TL AUPR.
- For LT+TL AUPR, `bxt-bgso-yr` and `bxt-gmo-yr` are the best models overall.

2.9.8.1 Setup

The algorithms being considered in this section are listed below, and their scoring results are shown by Figure 18.

- **`bxt-bgso-yr`, `bxt-gmo-yr`, `bxt-sq-yr`:** Bipartite Extra-Trees using the NRLMF model to impute missing annotations. The BGSO model implements our optimized method for growing the trees. GMO uses the global multi output strategy developed by 1. Both BGSO and GMO grows the trees to their maximum depth. SQ uses the weighted-neighbors prototype we developed in Section 2.4.3, and enforces at least 5 samples of each domain in each leaf. See Section 2.9.3 and Section 2.9.5 for more information.

- **NRLMF:** Neighborhood-Regularized Logistic Matrix Factorization, as proposed by 31 and described in Section A.2.2. See Section 2.9.5 for the hyperparameters we utilized.

- **RLS-Kron:** Kronecker Regularized Least Squares, as proposed by 30 and described in Section A.1. Each of the two input kernel matrices was taken as a linear combination of the similarity matrix and the gaussian interaction profiles (30). The weight of the similarity kernel in this combination (the α parameter (30)), was selected between the values $\{0.0, 0.1, 0.25, 0.5, 0.75, 0.9, 1.0\}$. The selection was performed in each fold by an internal 5 by 5 bipartite diagonal CV procedure (Section 2.7.2).

- **SLMO-RLS:** SLMO adaptation (Section 2.2.2) of Regularized Least Squares, as proposed by 30 as RLS-avg. Both the primary and secondary models were multi-output kernel Ridge regressors (96, p. 492-493). As RLS-Kron, the input features were linear combinations of similarity matrices and gaussian interactions profiles. The α parameter was selected in the same way as in RLS-Kron.

- **BLMNII-RLS:** Bipartite local models (70,95) with neighbor-based interaction-profile inferring (97). They use the SLMO strategy (Section 2.2.2), employing a weighted-neighbors technique for the primary estimators and kernel Ridge regression (96, p. 492-493) models as secondary estimators. As RLS-Kron, the input features were linear combinations of similarity matrices and gaussian interactions profiles (30). The α parameter was selected in the same way as in RLS-Kron.

- **BLMNII SVM:** The same as BLMNII RLS, but using support vector machines (98) as secondary estimators instead of regularized least squares.

- **DT-Hybrid:** Method proposed by 99 that combines the similarity kernels with network features of each domain calculated from the interaction matrix. A weighted-neighbors approach is then used to predict new interactions.

- **MLP:** Multi-layer perceptron (63) model adapted with the SGSO strategy (Section 2.2.1). We performed random undersampling of negative annotations so that the model was trained on equal number of negative and positive labels. The architecture was selected at each fold between four options: 5 hidden layers of 100 neurons; 10 hidden layers of 50 neurons; 5 hidden layers with [200, 100, 100, 100, 50] neurons; and 6 hidden layers of [1024, 512, 256, 128, 64, 32] neurons. The architecture selection in each fold was performed by an internal 5 by 5 bipartite diagonal CV procedure (Section 2.7.2). The activation function was always the rectified linear unit, and ADAM (100) was the chosen optimizer. Other parameters were kept as the defaults from the Scikit-learn library (101).

2.9.8.2 Discussion

`bxt-sq-yr` was the best ranked model in 11 out of the 16 test settings analysed. The exceptions were the four LT+TL AUPR configurations and TT AUPR 0%. In TT AUPR 0%, `bxt-sq-yr` was only behind SLMO rls, and among the three models that could not be statistically distinguished from SLMO rls (the others being RLS-Kron and `bxt-gmo-yr`). In LT+TL AUROC 0%, the comparison between `bxt-sq-yr` and the second best model, `bxt-gmo-yr`, was also not statistically significant. In the remaining 10 cases where `bxt-sq-yr` prevails, it was significantly better than all other learning algorithms analysed.

Regarding LT+TL AUPR, in ILR = 50 and ILR = 70 `bxt-bgso-yr` and `bxt-gmo-yr` significantly outperform the other models. In ILR = 0, `bxt-gmo-yr` was the best model and `bxt-bgso-yr` was the second best, but neither could be statistically distinguished from the third best model, `rls-kron`. In ILR = 90, `bxt-bgso-yr` was the best model and `bxt-gmo-yr` was the second best, but neither could be statistically distinguished from the third best model, RLS-Kron. Under ILR = 90, `bxt-bgso-yr` and `bxt-sq-yr` are shown to significantly outperform the others.

We notice that the highest ranked bipartite forests in each evaluation setting are always able to significantly surpass NRLMF alone. This demonstrates that the forests are able to capture different patterns than the NRLMF model, even though they are trained on NRLMF's outputs. If this result were to not hold, one could argue that the forests might be merely approximating the predictive function learned by NRLMF, rather than expanding on the information extracted in the matrix factorization step.

We also note that the competitiveness of the `rls-kron` model for TT AUPR 0% and LT+TL AUPR 0% is remarkable, since `rls-kron` has notably low training times in comparison to the other models. However, the algorithm seems to not perform as well when a larger number of positive annotations is missing. This can be seen when comparing those same test metrics under higher ILR: TT AUPR 70%, TT AUPR 90%, LT+TL AUPR 70% LT+TL AUPR 90%. Furthermore, `rls-kron` seems to be prioritized by AUPR in comparison to AUROC. This is especially suggested when observing the four TT AUROC, in which `rls-kron` performs poorly in comparison to the other cases. According to the discussion in Section 2.9.1, this could mean that `rls-kron` is mostly effective when considering a restricted number of top predictions.

Conversely, the performance of `bxt-sq-yr` for the LT+TL AUPR setting seems to improve relative to the other models as the number of missing annotations increases. A possible explanation comes from BXT SQ having higher generalization capabilities, as demonstrated in the TT AUROC setting. Notice that the LT+TL setting could possibly still benefit from some form of overfitting, since it considers instances that are present in the training set (Section 2.7.2). However, being able to generalize becomes more important as the number of missing annotations increases, since the model has to rely on less information to make predictions. And that could be why we see the `bxt-sq-yr` model being more valued in the LT+TL AUPR setting for higher ILR. Additionally, we argue that the AUPR metric is less sensitive than AUROC to this ability of discovering missing positives Section 2.9.1, which explains why AUROC does not show the same pattern as AUPR in the LT+TL setting.

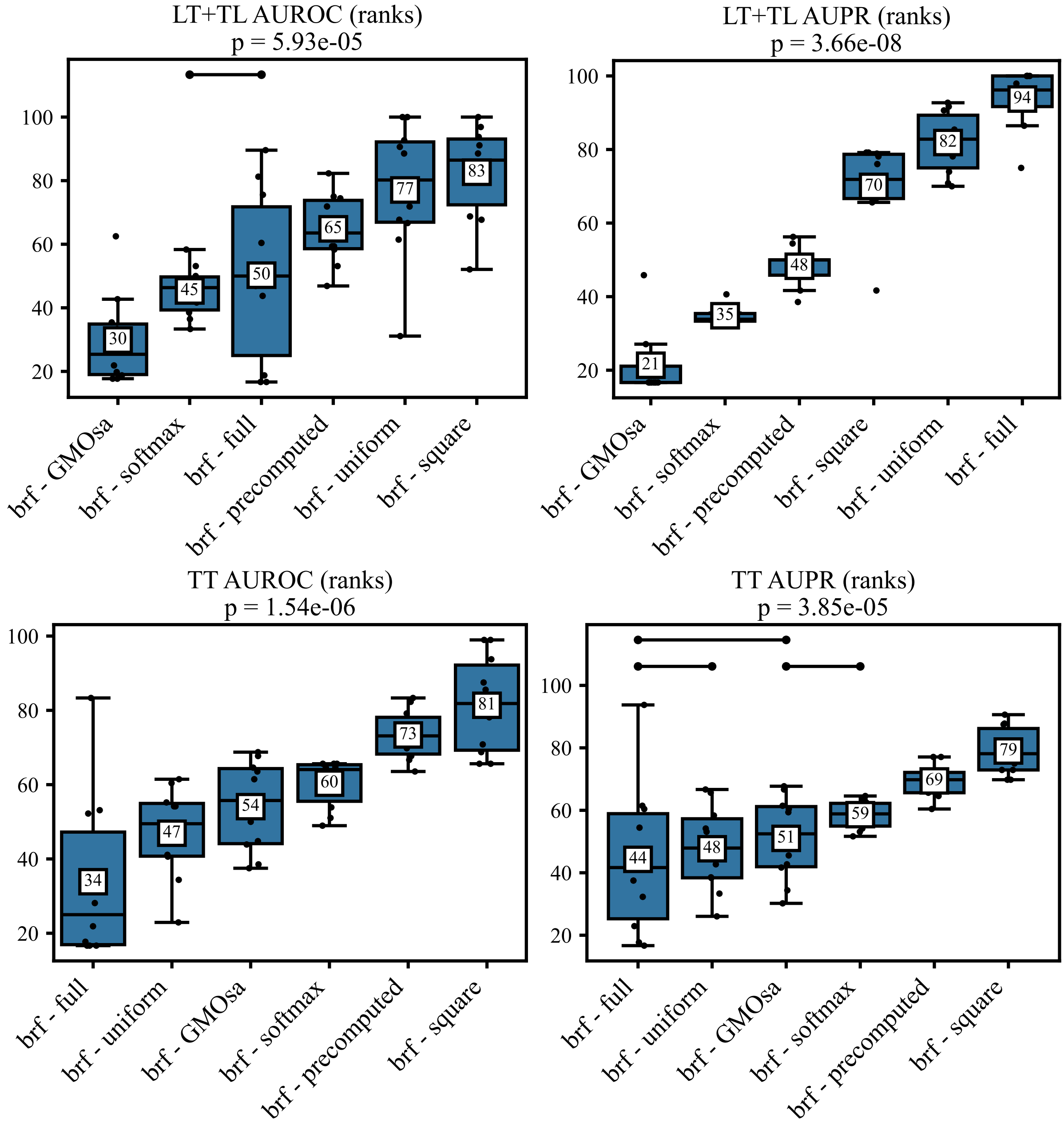

Figure 10 – Comparison of bipartite random forests for different prototype strategies. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.

Source: By the author.

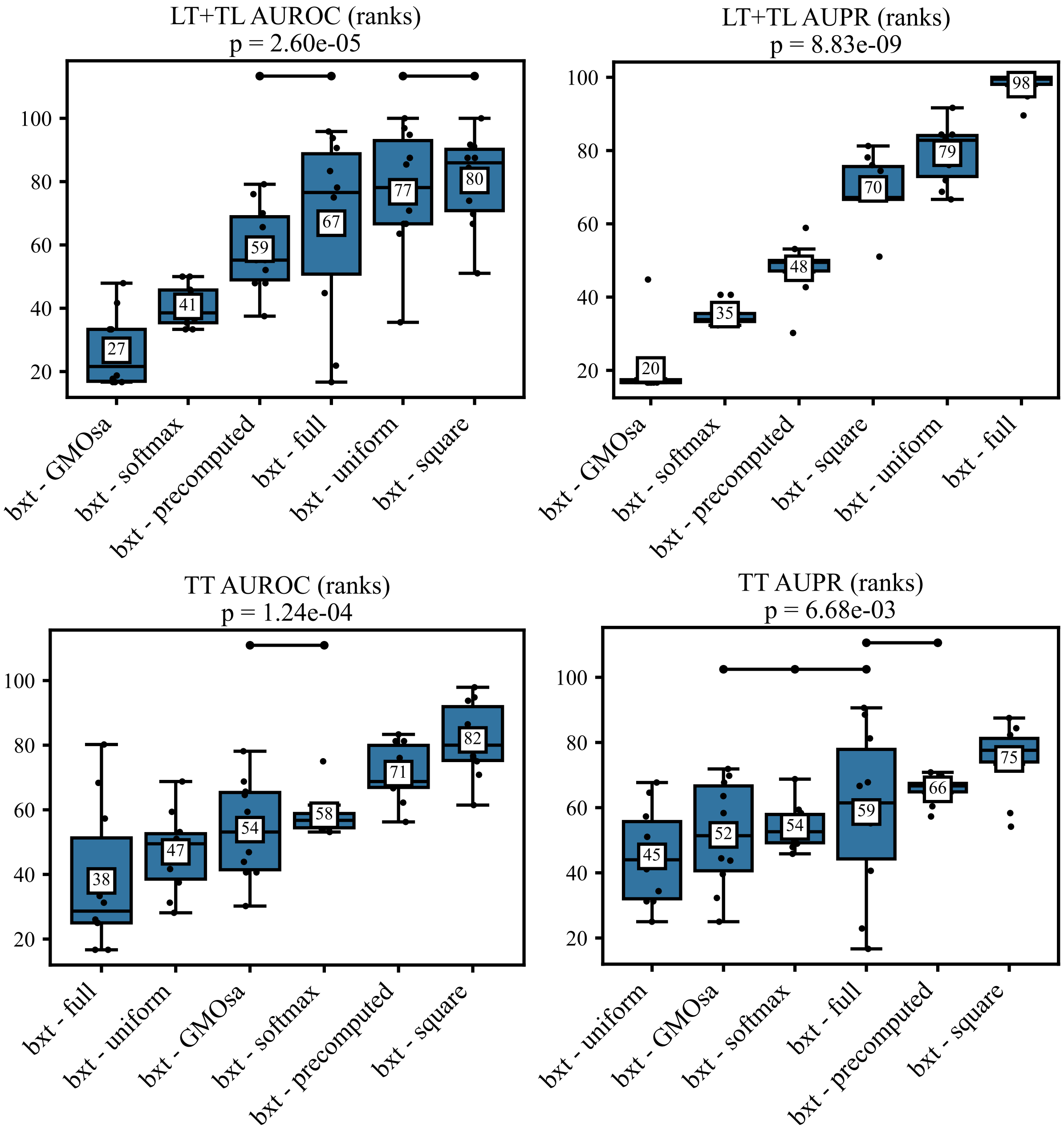

Figure 11 – Comparison of bipartite Extra-Trees for different prototype strategies. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author.

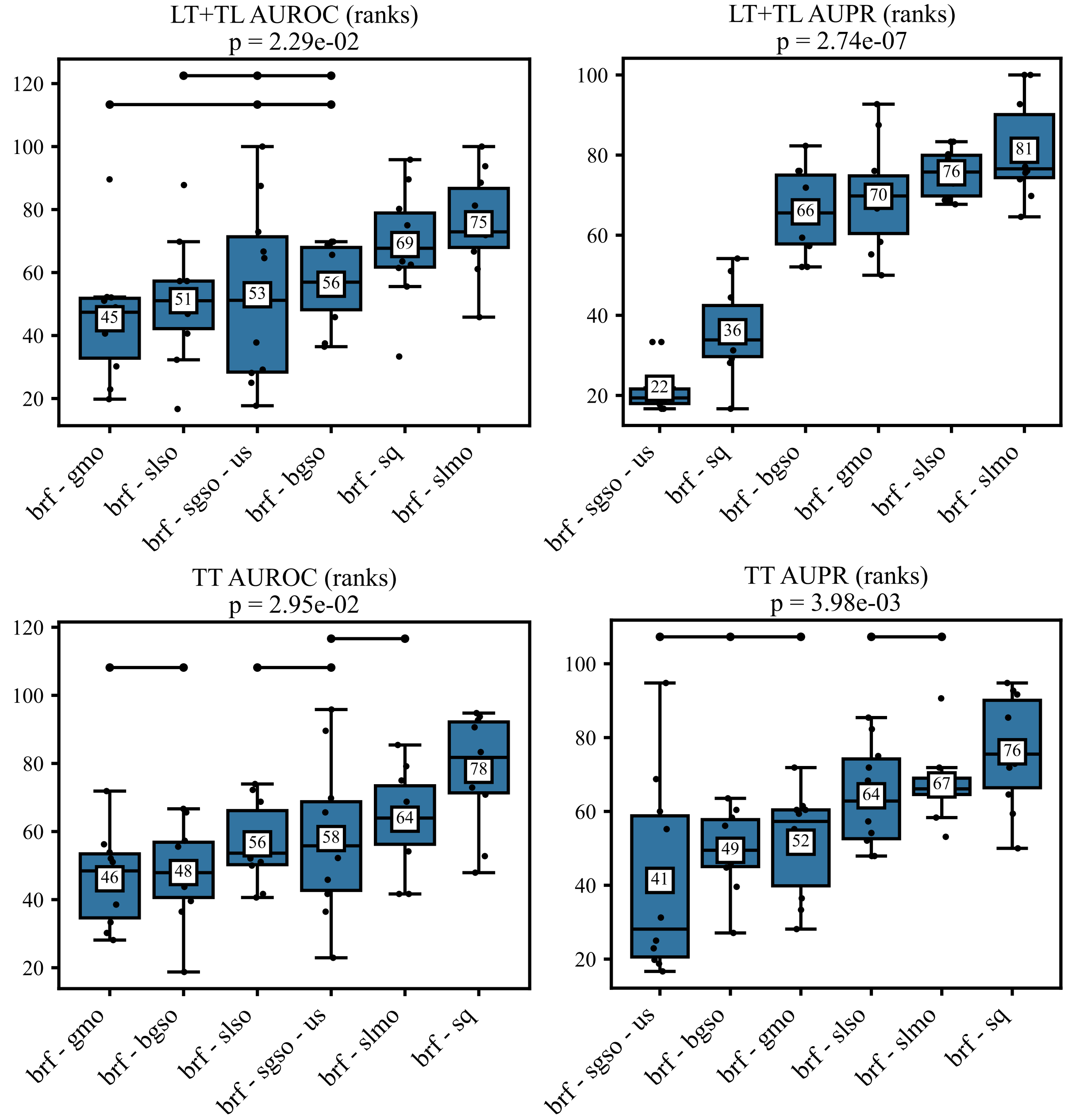

Figure 12 – Comparison of random forests under different adaptation strategies to bipartite interaction data. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.

Source: By the author.

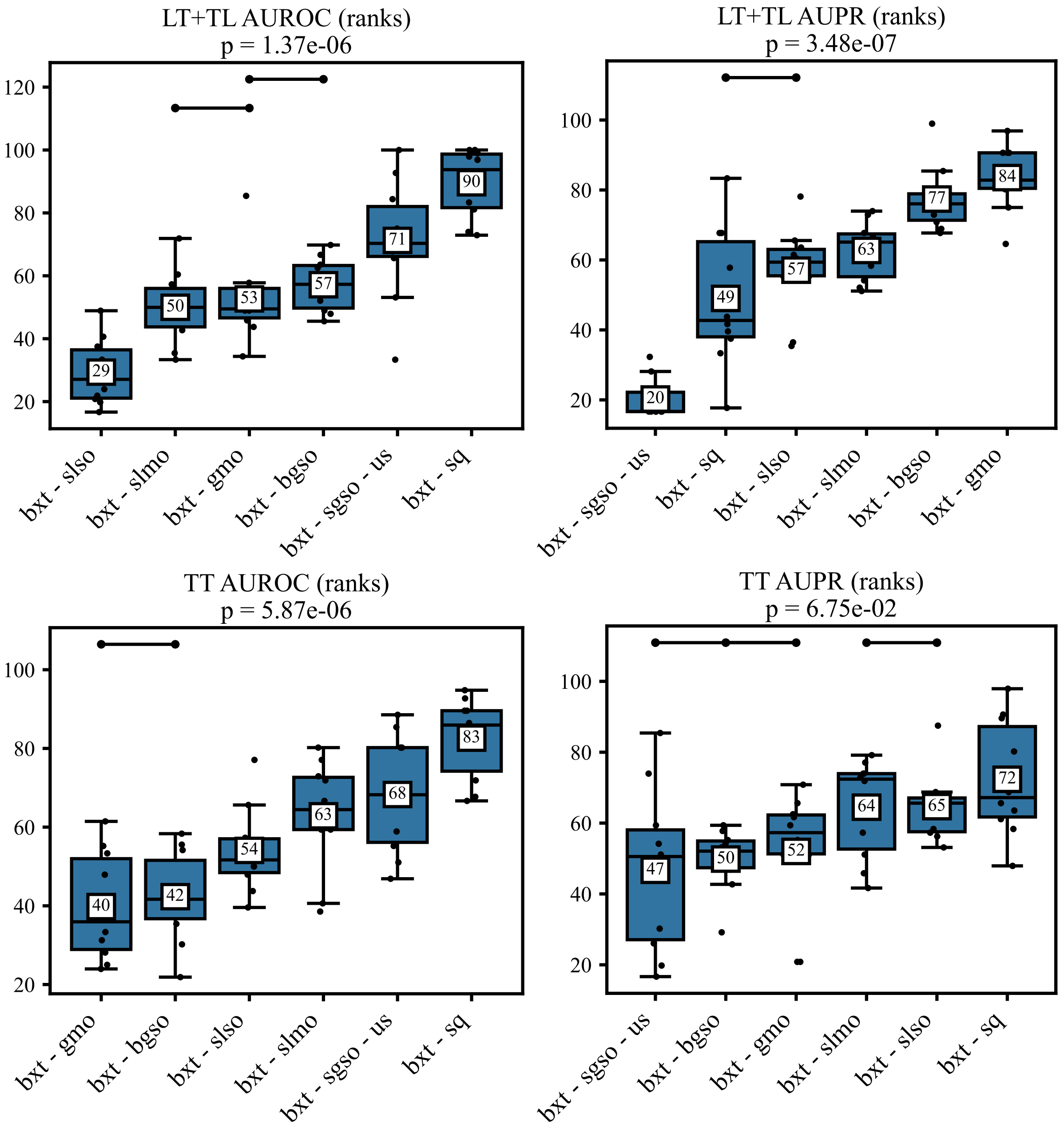

Figure 13 – Comparison of Extra-Trees under different adaptation strategies to bipartite interaction data. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.

Source: By the author.

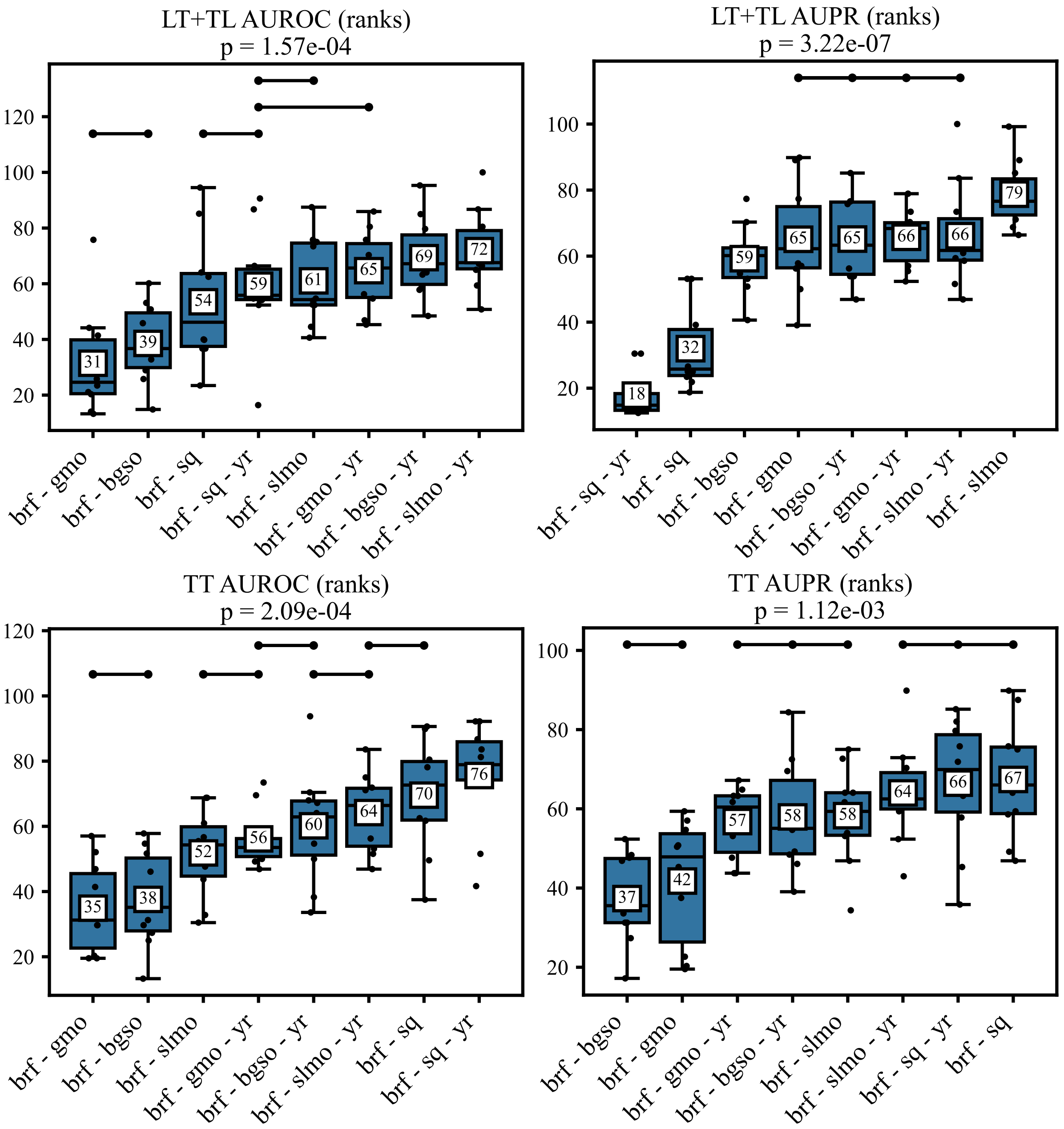

Figure 14 – Comparison of bipartite random forests with and without label imputation by NRLMF. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.

Source: By the author.

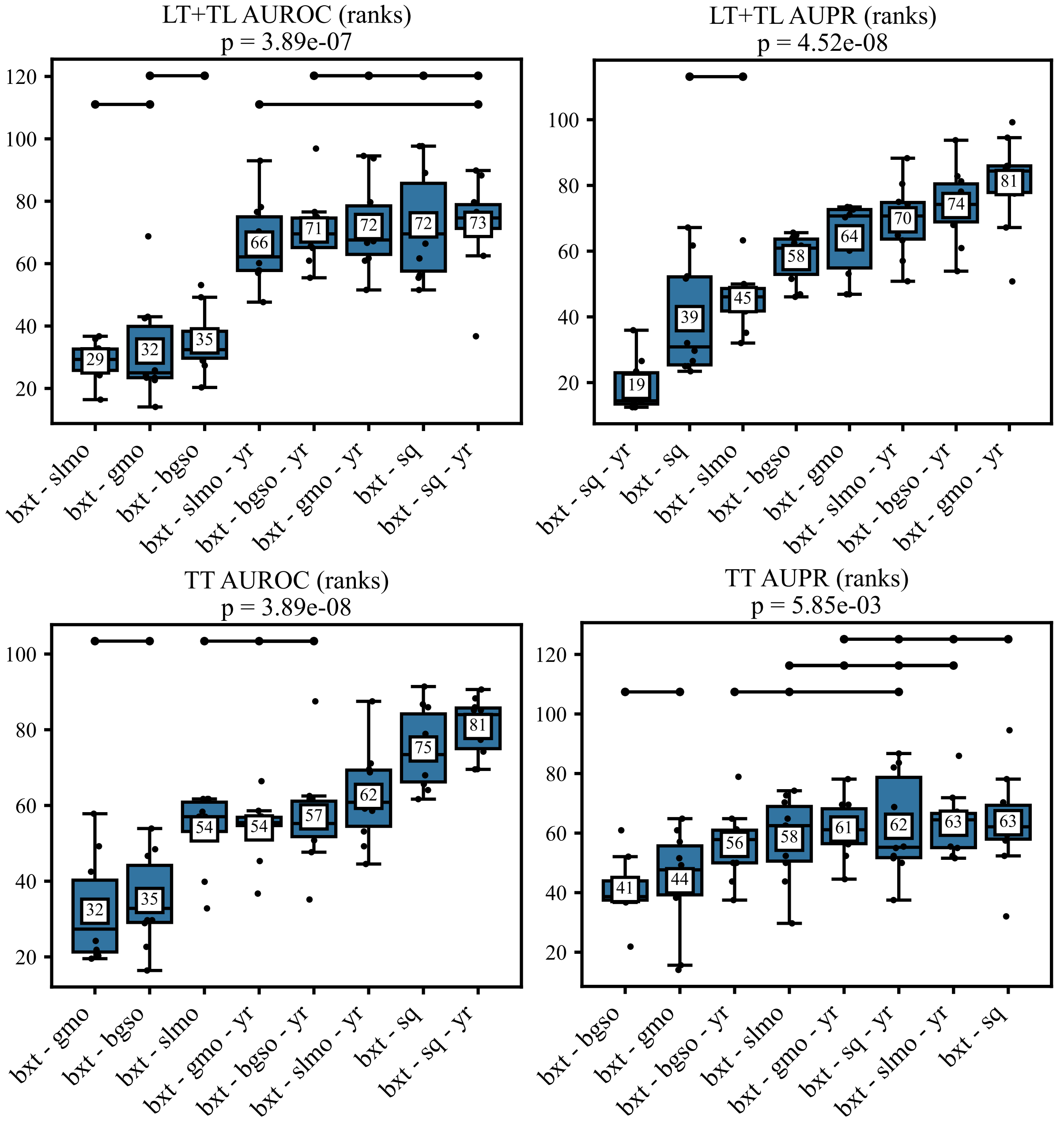

Figure 15 – Comparison of bipartite Extra-Trees with and without label imputation by neighborhood-regularized logistic matrix factorization. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author.

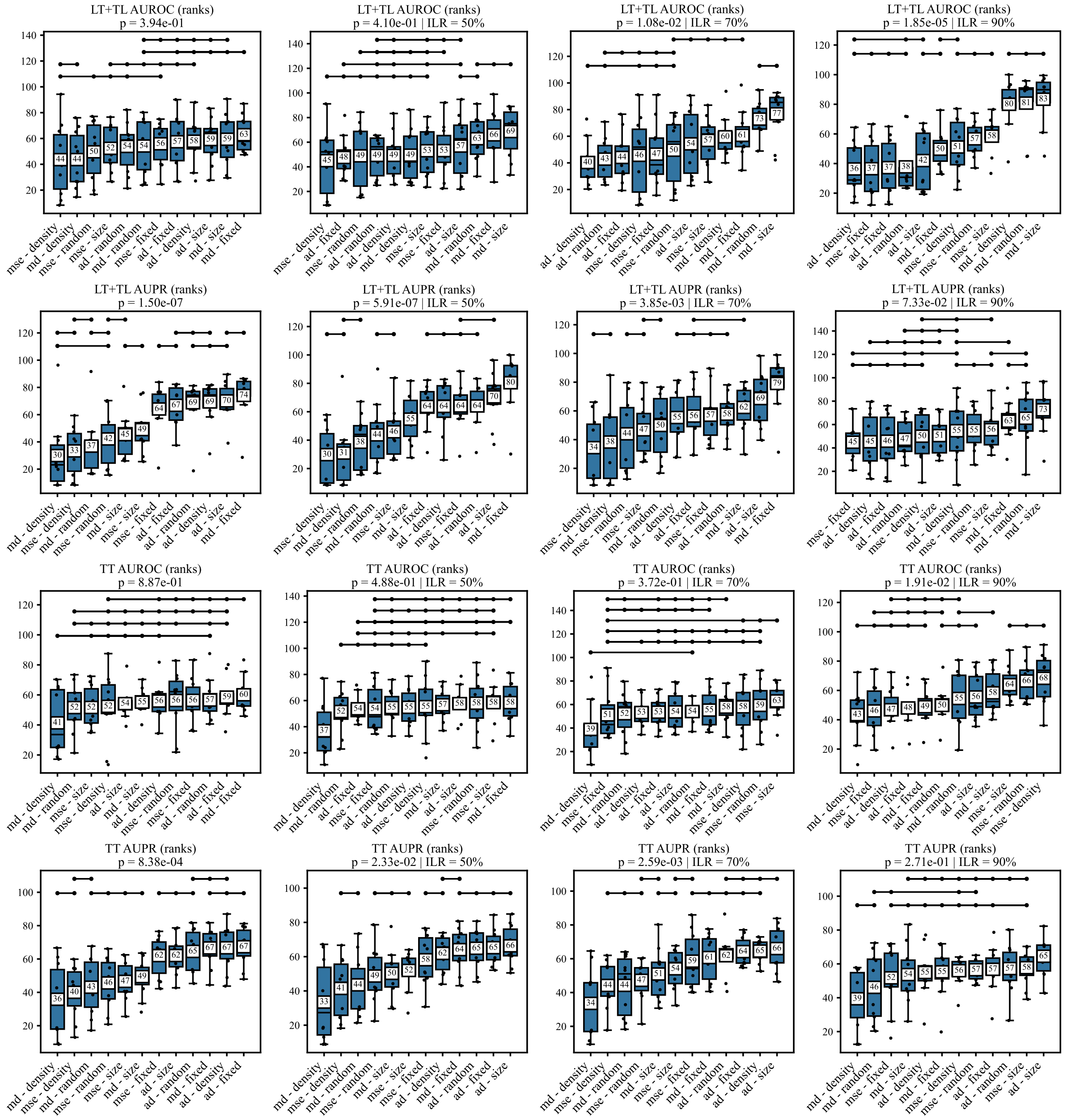

Figure 16 – Comparison of semi-supervised BGSO BXT with different strategies for determining the unsupervised impurity and the supervision balance. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author.

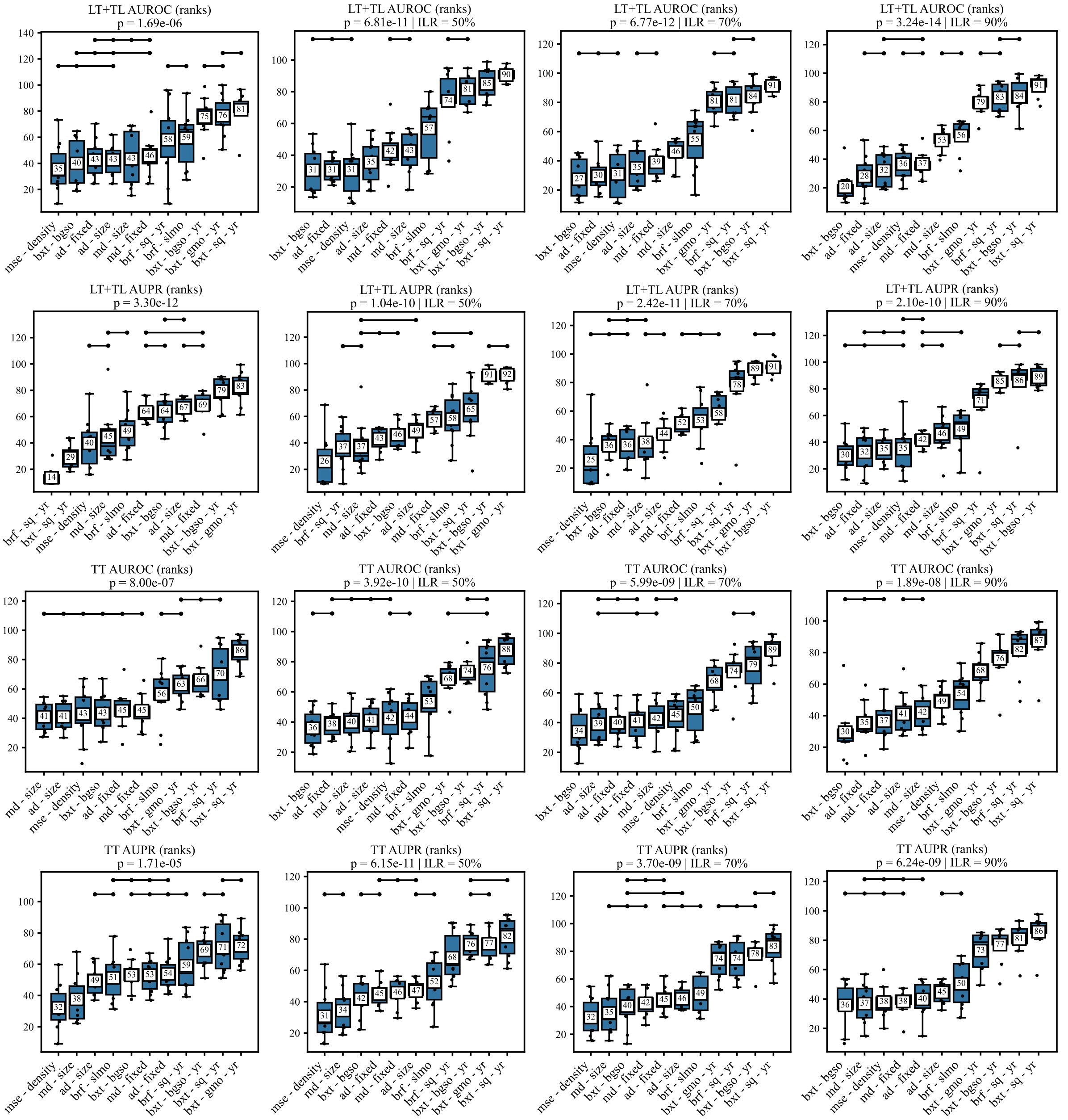

Figure 17 – Comparison of different bipartite forests. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure (4, 5) in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.
Source: By the author.

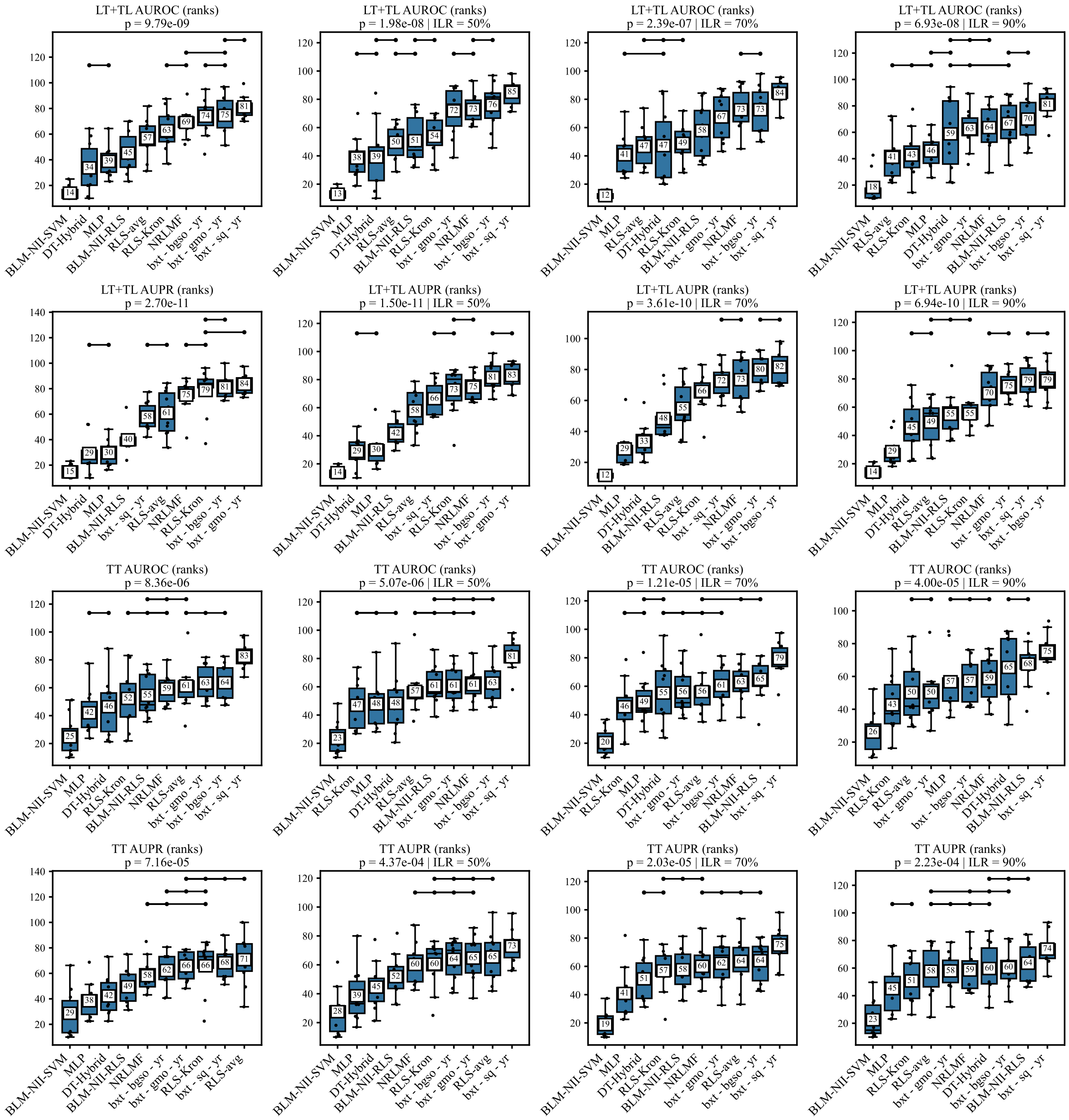

Figure 18 – Comparison of two proposed bipartite forests (`bxt-sq-yr` and `bxt-bgso-yr`) with several prominent models from the literature. An omnibus p-value is obtained through a Friedman test and indicated below the title of each subfigure. We then perform pairwise Wilcoxon rank-sum tests as a post-hoc analysis. Estimators that could *not* be significantly distinguished from each other ($p > 0.05$) are connected by crossbars above their respective boxes. The pairwise test results are corrected by the Benjamini-Hochberg procedure $(4, 5)$ in each subfigure. All pairwise comparisons are considered for the correction, even if not visible in the plot. See Section 2.8 for further description of the evaluation procedure.

Source: By the author.

3 CONCLUSION

In this chapter, we summarize the main findings of this work and discuss future research directions.

3.1 Main findings

We now list our final answers to the research questions proposed in Section 1.6.

1. **Can bipartite trees be faster?**

   Yes, we developed significantly faster-growing bipartite trees by exploiting the bipartite nature of the problem.

   We demonstrate both theoretically (Section 2.4.2) and empirically (Section 2.9.2) that our proposed bipartite tree algorithm achieves a $\log n$ improvement in training time complexity relative to its predecessors, with comparable predictive performance (Section 2.9.8).

2. **Are semi-supervised techniques beneficial for bipartite forests?**

   Yes. Overall, we show that both semi-supervised impurities and label imputation by matrix factorization improve the predictions of bipartite forests.

   For most scenarios, using our proposed semi-supervised impurity improves the predictive performance relative to the corresponding supervised model (Section 2.9.7). However, further investigation is needed to evaluate the benefits of other impurity functions. Additional experiments would also be interesting to investigate the effect of the methods that define the supervision balance parameter.

   In any case, the most significant improvements were achieved by reconstructing the interaction matrix. As proposed by (29), we employ neighborhood-regularized matrix factorization (31) (NRLMF) to impute positive annotations prior to training the forests, and show that this approach consistently improves the scores in almost all settings (Section 2.9.5 and Section 2.9.7).

3. **How do AUROC and AUPR differ in their assessment of model performance?**

   AUPR prioritizes a smaller number of highest-ranked interactions, while AUROC considers a larger number of both highest and lowest ranks.

   We show that AUPR is very sensitive to which interactions from the test set the estimator selects as the most likely to occur. Also, AUPR mostly disregards the interactions selected as unlikely by the estimator. On the other hand, AUROC equally considers both the highest and lowest ranked interactions, and is less influenced by the extremal ranks in comparison to AUPR (Section 2.9.1).

Therefore, we argue that AUROC should be generally favored for model comparison under positive-unlabeled settings. One should favor AUPR instead when the main goal is to select a small number of most likely interactions.

4. **How do bipartite forests compare with proficient models in the field?**

   The bipartite forests showed the best results in our experimental comparisons.

   Notably, the BXT GMO NRLMF model stood out in most test settings analysed, especially when predicting interactions between unknown instances (Section 2.9.8). This model employs NRLMF label imputation and our proposed prototype function, demonstrating the effectiveness of these techniques in improving generalization.

3.2 Future work

The present study opens several paths for future research. We enumerate some of them below.

1. **Adding noise to the impurity**

   It is possible that using completely random values for the unsupervised impurity could still result in improved performance for the semi-supervised forests. This is because the diversity introduced by the noisy impurity to the individual trees could be enough, in theory, to improve the performance of the ensemble.

   Hence, we propose adding noise to the supervised impurity and compare the results with supervised and semisupervised trees. The experiment could bring insights into the importance of different impurities in comparison with adding diversity to the trees.

2. **There is a lot of unsupervised impurities to be explored**

   The present work explored two options of unsupervised impurity functions, that are utilized for training semi-supervised decision trees. However, unsupervised decision trees are a vast topic to explore. Many other approaches could be brought to the bipartite setting, with potential benefits for the predictive performance and scalability of the forests.

   For instance, several works use a single feature column to determine an unsupervised impurity (102). Their reasoning is that the most significant change in impurity will result from the feature being used to select the split points. This feature is previously ordered by the algorithm. Therefore, other feature columns can be disregarded.

   Other unsupervised impurities (102) leverage not only the smoothness assumption but also the *cluster assumption*, which tells us to stablish decision borders preferably in low-density regions of the feature space (20, 41). These impurities then prioritize split points separating groups that are far from each other in the feature space, and not only compact (low variance) (102).

Other approaches (40) also use the structure of random trees to determine similarity values (42), obtaining a similarity matrix to various types of data.

3. **Semi-supervised impurities for all forest models**

   Only the bipartite global single-output ExtraTrees were trained with semi-supervised impurities. However, other models could benefit from the same approach, maybe even to a greater extent. Future investigations could explore the use of semi-supervised impurities with Random Forests and global multi-output forests. Another promising strategy would be to combine such impurities with label imputation by matrix factorization, since imputation by itself proved to be effective.

4. **Other label imputation approaches**

   As previous studies, we used neighborhood-regularized matrix factorization (NRLMF) to impute positive annotations before training bipartite forests. The results were promising, but many other methods for label imputation could be explored.

   The neighborhood regularization terms in Equation A.30 are not strictly necessary if the labels are used to train a second model. Their main importance is in inferring labels for new instances, which could be done solely be the downstream model in this case. More generally speaking, the step of label imputation can be purely transductive (20, 41), as no induction of a function $f : \mathbf{x} \mapsto y$ is necessary. Therefore, it is possible that simpler methods achieve similar results in less time. Examples of promising options to try would be logistic matrix factorization (26) and graph-based learning methods in general (20,41).

   Additionally, there have been further developments of NRLMF that could be explored. NRLMF-β (103) adds beta-distribution-rescoring to the outputs of NRLMF. Dual network integrated logistic matrix factorization (64, 65) uses a concept similar to NRLMF, but merging the similarity matrices with the interaction matrix before factorization.

   Even the same forest algorithms could be used both for label imputation and the final inference step. This would be a form of self-learning, where the model is trained with the labels it has predicted itself (41). In a similar fashion, co-training could be used, where models are trained in separate folds and then used to infer labels for the other models (41). Self-learning can also be taken one step further, building a cascade of sequential forests that progressively impute missing annotations (93).

5. **Other forest algorithms**

   There is a variety of tree-based algorithms that could be adapted to the bipartite setting but were not explored in this work. Some of them are: Gradient Boosting Machines (33,104), Rotation Forests (105), Fuzzy Random Forests (106), Oblique Random Forests (32) and Deep Forests (34).

6. **Other fields**

The present study concentrated on scenarios in which both instance domains have respective feature matrices. However, bipartite forests could also be extended to contexts where only one of the domains has side-features (dyadic prediction, multi-label learning, weak-label learning) or even to scenarios where both domains are featureless (collaborative filtering, recommendation systems). In these cases, features can be extracted from the interaction matrix itself, for instance, as gaussian interaction profiles (30).

For multi-output scenarios, bipartite forests can learn to partition the label matrix both horizontally and vertically, learning patterns that are specific for certain groups of outputs. This property is explored by (38) for single trees and hierarchical multi-label classification.

Weak-label learning (93, 107) could also be approached in a similar way. The difference from traditional multi-label learning is that negative annotations are not validated, similarly to the PU learning (44) setting of the present work. As such, the semi-supervised techniques we developed could be especially advantageous for weak-label learning.

7. **Explainable artificial intelligence**

   Decision Forests have remarkable interpretability properties that were not explored in the present work. Future studies could exploit explanation techniques (32, 36, 37) to provide new insights into the nature of each learning task. For instance, in mircroRNA-gene interaction prediction, it would be interesting to pinpoint specific positions or sequence motifs that are relevant for the interaction.

REFERENCES

1 PLIAKOS, K.; GEURTS, P.; VENS, C. Global multi-output decision trees for interaction prediction. **Machine Learning**, v. 107, p. 1257–1281, 2018.

2 GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. **Machine Learning**, v. 63, p. 3–42, 2006.

3 BREIMAN, L. *et al.* **Classification and regression trees**. New York: Routledge, 1984. ISBN 978-1-315-13947-0.

4 BENJAMINI, Y.; HOCHBERG, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. **Journal of the Royal Statistical Society Series B (Methodological)**, v. 57, n. 1, p. 289–300, 1995.

5 HAYNES, W. Benjamini–Hochberg method. *In*: DUBITZKY, W. *et al.* (ed.). **Encyclopedia of systems biology**. New York: Springer, 2013. p. 78–78. ISBN 978-1-4419-9863-7.

6 BAGHERIAN, M. *et al.* Machine learning approaches and databases for prediction of drug–target interaction: a survey paper. **Briefings in Bioinformatics**, v. 22, n. 1, p. 247–269, Jan. 2020.

7 CHEN, R. *et al.* Machine learning for drug-target interaction prediction. **Molecules**, v. 23, n. 9, p. 2208, Aug. 2018.

8 FAITH, J. J. *et al.* Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. **PLoS Biology**, v. 5, n. 1, p. e8, 2007.

9 LÜ, L. *et al.* Recommender systems. **Physics Reports**, v. 519, n. 1, p. 1–49, Oct. 2012.

10 ASRATIAN, A. S.; DENLEY, T. M.; HÄGGKVIST, R. **Bipartite graphs and their applications**. Cambridge: Cambridge University Press, 1998. (Cambridge tracts in mathematics, v. 131).

11 EZZAT, A. *et al.* Computational prediction of drug–target interactions using chemogenomic approaches: an empirical survey. **Briefings in Bioinformatics**, v. 20, n. 4, p. 1337–1357, Jul. 2019.

12 LÜ, L.; ZHOU, T. Link prediction in complex networks: a survey. **Physica A**: statistical mechanics and its applications, v. 390, n. 6, p. 1150–1170, Mar. 2011.

13 ZHOU, T. Progresses and challenges in link prediction. **Iscience**, v. 24, n. 11, 2021.

14 MENON, A. K.; ELKAN, C. A log-linear model with latent leatures for dyadic prediction. *In*: INTERNATIONAL CONFERENCE ON DATA MINING, 10, 2010, Washington. **Proceedings [...]**. Washington: IEEE Press, 2010. p. 364–373.

15 PAHIKKALA, T. *et al.* A two-step learning approach for solving full and almost full cold start problems in dyadic prediction. *In*: CALDERS, T. *et al.* (ed.). **Machine learning and knowledge discovery in databases**. Berlin: Springer, 2014. p. 517–532. ISBN 978-3-662-44851-9.

16 JIN, B. *et al.* Multitask dyadic prediction and its application in prediction of adverse drug-drug interaction. *In*: AAAI CONFERENCE ON ARTIFICIAL INTELIGENCE, 31, 2017, Vancouver. **Proceedings [...]**. Vancouver: AAAI Press, 2017.

17 SCHRYNEMACKERS, M. *et al.* Classifying pairs with trees for supervised biological network inference. **Molecular BioSystems**, v. 11, n. 8, p. 2116–2125, 2015.

18 PLIAKOS, K.; VENS, C. Network inference with ensembles of bi-clustering trees. **BMC Bioinformatics**, v. 20, p. 1–12, 2019.

19 PAHIKKALA, T. *et al.* Toward more realistic drug-target interaction predictions. **Briefings in Bioinformatics**, v. 16, n. 2, p. 325–337, 2015.

20 CHAPELLE, O.; SCHÖLKOPF, B.; ZIEN, A. (ed.). **Semi-supervised learning**. Cambridge: MIT Press, 2006. (Adaptive computation and machine learning). ISBN 978-0-262-03358-9.

21 DAVIS, J.; GOADRICH, M. The relationship between precision-recall and ROC curves. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 23, 2006, New York. **Proceedings [...]**. New York: ACM, 2006. p. 233–240. ISBN 978-1-59593-383-6.

22 HAND, D. J. Measuring classifier performance: a coherent alternative to the area under the ROC curve. **Machine Learning**, v. 77, n. 1, p. 103–123, Oct. 2009.

23 OZENNE, B.; SUBTIL, F.; Maucort-Boulch, D. The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases. **Journal of Clinical Epidemiology**, v. 68, n. 8, p. 855–859, Aug. 2015.

24 FLACH, P.; KULL, M. Precision-recall-gain curves: PR analysis done right. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 28, 2015, Montreal. **Proceedings [...]**. Montreal: Curran Associates, Inc., 2015.

25 HE, H.; GARCIA, E. A. Learning from imbalanced data. **IEEE Transactions on Knowledge and Data Engineering**, v. 21, n. 9, p. 1263–1284, Sept. 2009.

26 JOHNSON, C. C. Logistic matrix factorization for implicit feedback data. **Advances in Neural Information Processing Systems**, v. 27, n. 78, p. 1–9, 2014.

27 HAO, M.; BRYANT, S. H.; WANG, Y. Open-source chemogenomic data-driven algorithms for predicting drug–target interactions. **Briefings in Bioinformatics**, v. 20, n. 4, p. 1465–1474, Jul. 2019.

28 YU, D. *et al.* FPSC-DTI: drug–target interaction prediction based on feature projection fuzzy classification and super cluster fusion. **Molecular Omics**, v. 16, n. 6, p. 583–591, Dec. 2020.

29 PLIAKOS, K.; VENS, C. Drug-target interaction prediction with tree-ensemble learning and output space reconstruction. **BMC Bioinformatics**, v. 21, p. 1–11, 2020.

30 LAARHOVEN, T. V.; NABUURS, S. B.; MARCHIORI, E. Gaussian interaction profile kernels for predicting drug–target interaction. **Bioinformatics**, v. 27, n. 21, p. 3036–3043, 2011.

31 LIU, Y. *et al.* Neighborhood regularized logistic matrix factorization for drug-target interaction prediction. **PLoS Computational Biology**, v. 12, n. 2, p. e1004760, Feb. 2016.

32 BREIMAN, L. Random forests. **Machine Learning**, v. 45, p. 5–32, 2001.

33 CHEN, T.; GUESTRIN, C. XGBoost: a scalable tree boosting system. *In*: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 22, 2016, New York. **Proceedings [...]**. New York: ACM/SIGKDD, 2016. p. 785–794. ISBN 978-1-4503-4232-2.

34 ZHOU, Z.-H.; FENG, J. Deep forest. **National Science Review**, v. 6, n. 1, p. 74–86, Jan. 2019.

35 GRINSZTAJN, L.; OYALLON, E.; VAROQUAUX, G. Why do tree-based models still outperform deep learning on typical tabular data? **Advances in Neural Information Processing Systems**, v. 35, p. 507–520, Dec. 2022.

36 LUNDBERG, S. M. *et al.* **Explainable AI for trees**: from local explanations to global understanding. 2019. Available at: http://arxiv.org/abs/1905.04610. Accessible at: 15 Mar. 2024.

37 AGARWAL, A. *et al.* **MDI+**: a flexible random forest-based feature importance framework. 2023. Available at: http://arxiv.org/abs/2307.01932. Accessible at: 13 Jan. 2024.

38 SANTOS, B. Z. *et al.* Predictive bi-clustering trees for hierarchical multi-label classification. *In*: HUTTER, F. *et al.* (ed.). **Machine learning and knowledge discovery in databases**. Cham: Springer International Publishing, 2021. p. 701–718. ISBN 978-3-030-67664-3.

39 LEVATIĆ, J. *et al.* Semi-supervised classification trees. **Journal of Intelligent Information Systems**, v. 49, p. 461–486, 2017.

40 ADIYEKE, E.; BAYDOĞAN, M. G. Semi-supervised extensions of multi-task tree ensembles. **Pattern Recognition**, v. 123, p. 108393, Mar. 2022.

41 ENGELEN, J. E. V.; HOOS, H. H. A survey on semi-supervised learning. **Machine Learning**, v. 109, n. 2, p. 373–440, Feb. 2020.

42 LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. *In*: INTERNATIONAL CONFERENCE ON DATA MINING, 8, 2008, Washington. **Proceedings [...]**. Washington, 2008. p. 413–422.

43 ALVES, A.; ILIDIO, P.; CERRI, R. Semi-supervised hybrid predictive bi-clustering trees for drug-target interaction prediction. *In*: SYMPOSIUM ON APPLIED COMPUTING, 38, 2023, Tallinn. **Proceedings [...]**. Tallinn: ACM/SIGAPP, 2023. p. 1163–1170.

44 BEKKER, J.; DAVIS, J. Learning from positive and unlabeled data: a survey. **Machine Learning**, v. 109, n. 4, p. 719–760, Apr. 2020.

45 VERT, J. P. **Reconstruction of biological networks by supervised machine learning approaches**. 2008. Available at: http://arxiv.org/abs/0806.0215. Accessible at: 29 Aug. 2023.

46 ÖZTÜRK, H.; ÖZGÜR, A.; OZKIRIMLI, E. DeepDTA: deep drug–target binding affinity prediction. **Bioinformatics**, v. 34, n. 17, p. i821–i829, 2018.

47 HUANG, K. *et al.* MolTrans: molecular interaction transformer for drug–target interaction prediction. **Bioinformatics**, v. 37, n. 6, p. 830–836, Oct. 2020.

48 QUINLAN, J. R. Induction of decision trees. **Machine Learning**, v. 1, n. 1, p. 81–106, Mar. 1986.

49 QUINLAN, J. R. **C4. 5**: programs for machine learning. Elsevier, 2014.

50 CORMEN, T. H. *et al.* **Introduction to algorithms**. 4th ed. Cambridge: MIT Press, 2022. ISBN 978-0-262-04630-5.

51 DIETTERICH, T. G. Ensemble methods in machine learning. *In*: GOOS, G.; HARTMANIS, J.; LEEUWEN, J. V. (ed.). **Multiple classifier systems**. Heidelberg: Springer, 2000, (Lecture notes in computer science, v. 1857). p. 1–15. ISBN 978-3-540-67704-8 978-3-540-45014-6.

52 POLIKAR, R. Ensemble based systems in decision making. **IEEE Circuits and Systems Magazine**, v. 6, n. 3, p. 21–45, 2006.

53 HYAFIL, L.; RIVEST, R. L. Constructing optimal binary decision trees is NP-complete. **Information Processing Letters**, v. 5, n. 1, p. 15–17, May 1976.

54 BREIMAN, L. Bagging predictors. **Machine Learning**, v. 24, n. 2, p. 123–140, Aug. 1996.

55 AMIT, Y.; GEMAN, D. Shape quantization and recognition with randomized trees. **Neural Computation**, v. 9, n. 7, p. 1545–1588, Oct. 1997.

56 HO, T. K. The random subspace method for constructing decision forests. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 20, n. 8, p. 832–844, Aug./1998.

57 SAGI, O.; ROKACH, L. Ensemble learning: a survey. **WIREs Data Mining and Knowledge Discovery**, v. 8, n. 4, p. e1249, Jul. 2018.

58 FAWAGREH, K.; GABER, M. M.; ELYAN, E. Random forests: from early developments to recent advancements. **Systems Science & Control Engineering**, v. 2, n. 1, p. 602–609, Dec. 2014.

59 AMASYALI, M. F.; ERSOY, O. K. Comparison of single and ensemble classifiers in terms of accuracy and execution time. *In*: INTERNATIONAL SYMPOSIUM ON INNOVATIONS IN INTELLIGENT SYSTEMS AND APPLICATIONS (INISTA), 2011, Istanbul. **Proceedings [...]**. Istanbul: IEEE, 2011. p. 470–474. ISBN 978-1-61284-919-5.

60 LIU, H. *et al.* LPI-NRLMF: lncRNA-protein interaction prediction by neighborhood regularized logistic matrix factorization. **Oncotarget**, v. 8, n. 61, p. 103975, 2017.

61 HE, T. *et al.* SimBoost: a read-across approach for predicting drug–target binding affinities using gradient boosting machines. **Journal of Cheminformatics**, v. 9, n. 1, p. 1–14, 2017.

62 ZHU, X.; GOLDBERG, A. B. **Introduction to semi-supervised learning**. Switzerland: Springer Nature, 2022.

63 HASTIE, T.; FRIEDMAN, J.; TIBSHIRANI, R. **The elements of statistical learning**. New York: Springer, 2001. (Springer series in statistics). ISBN 978-1-4899-0519-2 978-0-387-21606-5.

64 HAO, M.; BRYANT, S. H.; WANG, Y. Predicting drug-target interactions by dual-network integrated logistic matrix factorization. **Scientific Reports**, v. 7, n. 1, p. 40376, 2017.

65 LI, Y.; LI, J.; BIAN, N. DNILMF-LDA: prediction of lncRNA-disease associations by dual-network integrated logistic matrix factorization and Bayesian optimization. **Genes**, v. 10, n. 8, p. 608, 2019.

66 POWERS, D. M. W. **Evaluation**: from precision, recall and F-measure to ROC, informedness, markedness and correlation. 2020. Available at: http://arxiv.org/abs/2010.16061. Accessible at: 15 Mar. 2024.

67 SAITO, T.; REHMSMEIER, M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. **PLOS ONE**, v. 10, n. 3, p. e0118432, Mar. 2015.

68 FERNÁNDEZ, A. *et al.* **Learning from imbalanced data sets**. Cham: Springer International Publishing, 2018. ISBN 978-3-319-98073-7 978-3-319-98074-4.

69 HU, Y.; KOREN, Y.; VOLINSKY, C. Collaborative filtering for implicit feedback datasets. *In*: INTERNATIONAL CONFERENCE ON DATA MINING, 8, 2008, Pisa. **Proceedings [...]**. Pisa: IEEE, 2008. p. 263–272.

70 YAMANISHI, Y. *et al.* Prediction of drug–target interaction networks from the integration of chemical and genomic spaces. **Bioinformatics**, v. 24, n. 13, p. i232–i240, 2008.

71 MACISAAC, K. D. *et al.* An improved map of conserved regulatory sites for Saccharomyces cerevisiae. **BMC Bioinformatics**, v. 7, n. 1, p. 1–14, 2006.

72 HUGHES, T. R. *et al.* Functional discovery via a compendium of expression profiles. **Cell**, v. 102, n. 1, p. 109–126, 2000.

73 HU, Z.; KILLION, P. J.; IYER, V. R. Genetic reconstruction of a functional transcriptional regulatory network. **Nature Genetics**, v. 39, n. 5, p. 683–687, 2007.

74 CHUA, G. *et al.* Identifying transcription factor functions and targets by phenotypic activation. **Proceedings of the National Academy of Sciences**, v. 103, n. 32, p. 12045–12050, 2006.

75 DAVIS, M. I. *et al.* Comprehensive analysis of kinase inhibitor selectivity. **Nature Biotechnology**, v. 29, n. 11, p. 1046–1051, 2011.

76 HUANG, K. *et al.* DeepPurpose: a deep learning library for drug–target interaction prediction. **Bioinformatics**, v. 36, n. 22-23, p. 5545–5547, 2020.

77 TANG, J. *et al.* Making sense of large-scale kinase inhibitor bioactivity data sets: a comparative and integrative analysis. **Journal of Chemical Information and Modeling**, v. 54, n. 3, p. 735–743, 2014.

78 WU, T. *et al.* NPInter: the noncoding RNAs and protein related biomacromolecules interaction database. **Nucleic Acids Research**, v. 34, n. suppl_1, p. D150–D152, 2006.

79 TENG, X. *et al.* NPInter v4. 0: an integrated database of ncRNA interactions. **Nucleic Acids Research**, v. 48, n. D1, p. D160–D165, 2020.

80 HSU, S.-D. *et al.* miRTarBase: a database curates experimentally validated microRNA–target interactions. **Nucleic Acids Research**, v. 39, n. suppl_1, p. D163–D169, 2011.

81 HUANG, H.-Y. *et al.* miRTarBase update 2022: an informative resource for experimentally validated miRNA–target interactions. **Nucleic Acids Research**, v. 50, n. D1, p. D222–D230, 2022.

82 BROHÉE, S. *et al.* Unraveling networks of co-regulated genes on the sole basis of genome sequences. **Nucleic Acids Research**, v. 39, n. 15, p. 6340–6358, 2011.

83 ROGERS, D.; BROWN, R. D.; HAHN, M. Using extended-connectivity fingerprints with Laplacian-modified Bayesian analysis in high-throughput screening follow-up. **Journal of Biomolecular Screening**, v. 10, n. 7, p. 682–686, 2005.

84 LANDRUM, G. *et al.* **Rdkit/rdkit**: 2023_03_2 (Q1 2023) Release. 2023. Zenodo. Available at: https://doi.org/10.5281/zenodo.8053810.

85 GRIFFITHS-JONES,, S. *et al.* miRBase: microRNA sequences, targets and gene nomenclature. **Nucleic Acids Research**, v. 34, n. suppl_1, p. D140–D144, 2006.

86 FRANKISH, A. *et al.* GENCODE 2021. **Nucleic Acids Research**, v. 49, n. D1, p. D916–D923, 2021.

87 COCK, P. J. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. **Bioinformatics**, v. 25, n. 11, p. 1422, 2009.

88 LIU, C. *et al.* NONCODE: an integrated knowledge database of non-coding RNAs. **Nucleic Acids Research**, v. 33, n. suppl_1, p. D112–D115, 2005.

89 ZHAO, Y. *et al.* NONCODE 2016: an informative and valuable data source of long non-coding RNAs. **Nucleic Acids Research**, v. 44, n. D1, p. D203–D208, 2016.

90 CONSORTIUM, U. UniProt: a worldwide hub of protein knowledge. **Nucleic Acids Research**, v. 47, n. D1, p. D506–D515, 2019.

91 DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **Journal of Machine Learning Research**, v. 7, p. 1–30, 2006.

92 BENAVOLI, A.; CORANI, G.; MANGILI, F. Should we really use post-hoc tests based on mean-ranks? **Journal of Machine Learning Research**, v. 17, n. 5, p. 1–10, 2016.

93 WANG, Q.-W.; YANG, L.; LI, Y.-F. Learning from weak-label data: a deep forest expedition. *In*: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 34, 2020, New York. **Proceedings [...]**. New York, 2020. p. 6251–6258.

94 KORNBROT, D. Point biserial correlation. *In*: **Wiley StatsRef**: statistics reference online. Hatfield: John Wiley & Sons, Ltd, 2014. ISBN 978-1-118-44511-2.

95 BLEAKLEY, K.; YAMANISHI, Y. Supervised prediction of drug–target interactions using bipartite local models. **Bioinformatics**, v. 25, n. 18, p. 2397–2403, Jul. 2009.

96 MURPHY, K. P. **Machine learning**: a probabilistic perspective. Cambridge: MIT Press, 2012. (Adaptive computation and machine learning series). ISBN 978-0-262-01802-9.

97 MEI, J.-P. *et al.* Drug–target interaction prediction by learning from local information and neighbors. **Bioinformatics**, v. 29, n. 2, p. 238–245, 2013.

98 CRAMMER, K.; SINGER, Y. On the algorithmic implementation of multiclass kernel-based vector machines. **Journal of Machine Learning Research**, v. 2, n. Dec, p. 265–292, 2001.

99 ALAIMO, S. *et al.* Drug–target interaction prediction through domain-tuned network-based inference. **Bioinformatics**, v. 29, n. 16, p. 2004–2008, 2013.

100 KINGMA, D. P.; BA, J. **Adam**: a method for stochastic optimization. 2017. Available at: http://arxiv.org/abs/1412.6980. Accessible at: 16 Mar. 2024.

101 PEDREGOSA, F. *et al.* Scikit-learn: machine learning in Python. **Journal of Machine Learning Research**, v. 12, n. 85, p. 2825–2830, 2011.

102 Loyola-Gonzalez, O. *et al.* An explainable artificial intelligence model for clustering numerical databases. **IEEE Access**, v. 8, p. 52370–52384, 2020.

103 BAN, T.; OHUE, M.; AKIYAMA, Y. NRLMF\upbeta: beta-distribution-rescored neighborhood regularized logistic matrix factorization for improving the performance of drug–target interaction prediction. **Biochemistry and Biophysics Reports**, v. 18, p. 100615, Jul. 2019.

104 NATEKIN, A.; KNOLL, A. Gradient boosting machines, a tutorial. **Frontiers in Neurorobotics**, v. 7, Dec. 2013.

105 RODRIGUEZ, J. J.; KUNCHEVA, L. I.; ALONSO, C. J. Rotation forest: a new classifier ensemble method. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 28, n. 10, p. 1619–1630, 2006.

106 BONISSONE, P. *et al.* A fuzzy random forest. **International Journal of Approximate Reasoning**, v. 51, n. 7, p. 729–747, Sept. 2010.

107 ZHOU, Z.-H. A brief introduction to weakly supervised learning. **National Science Review**, v. 5, n. 1, p. 44–53, 2018.

108 SCHAFER, R. D. **An introduction to nonassociative algebras**. New York: Academic Press, 1966.

109 LIU, H. *et al.* Predicting lncRNA–miRNA interactions based on logistic matrix factorization with neighborhood regularized. **Knowledge-Based Systems**, v. 191, p. 105261, 2020.

110 RAFAILIDIS, D.; NANOPOULOS, A. Modeling users preference dynamics and side information in recommender systems. **IEEE Transactions on Systems, Man, and Cybernetics**: systems, v. 46, n. 6, p. 782–792, Jun. 2016.

111 DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. 61, p. 2121–2159, 2011.

APPENDIX

APPENDIX A – OTHER ESTIMATOR-CENTRIC STRATEGIES

This section presents two other methods for optimizing estimators for interaction prediction tasks. Like bipartite trees (Section 2.4), these methods focus on adapting the learning algorithm in a more fundamental level than data-centric approaches (Section 2.2). The first method is an adaptation of linear models to the bipartite setting, named RLS-Kron (30). It explores properties of the Kronecker product to build an estimator on a large global kernel matrix, without in fact having to calculate such matrix. The second method adapts matrix factorization approaches to take similarity side-features into account. Named Neighborhood-Regularized Logistic Matrix Factorization (NRLMF), this technique allows obtaining latent features for unseen instances, which is commonly not possible with traditional matrix factorization.

A.1 Linear models

One of the simplest approaches to learning problems in general is to assume a linear relationship between the input features and output labels. Formally, for the non-bipartite case, one assumes that the training output matrix Y can be approximated as follows:

$$\hat{Y} = XW \tag{A.1}$$

in which W is a matrix representing the set of parameters to be learned. To determine W, the mean squared error (MSE) is usually defined as the loss function to be minimized, to which we add an extra regularization term controlled by the hyperparameter α:

$$\mathcal{J} = \frac{1}{2}\|Y - \hat{Y}\|^2 + \frac{\alpha}{2}\|W\|^2 = \frac{1}{2}\|Y - XW\|^2 + \frac{\alpha}{2}\|W\|^2 \tag{A.2}$$

An analytical solution for W can be obtained by taking the derivative of $\mathcal{J}$ with respect to W and setting it to zero:

$$\frac{\partial \mathcal{J}}{\partial W} = 0 = X^\intercal(XW - Y) + \alpha W \implies W = (X^\intercal X + \alpha I)^{-1}X^\intercal Y \tag{A.3}$$

There are scenarios, however, where the specific values of X are less interesting than the pairwise similarities between them. In those settings, while X may not be directly available, we do have access to similarity matrices S (also called *kernel* matrices) in which $S^{[ij]}$ designates a similarity score between $X^{[i]}$ and $X^{[j]}$. Rather than simply considering S in the same way we would treat X in linear regression, we could instead employ the *kernel trick* (96): replacing the $XX^\intercal$ terms in the above equations by S. In this case, we are assuming that similarities $S^{[ij]}$ represent the internal product of the vectors $X^{[i]}$ and $X^{[j]}$ in some feature space, which is based on the intuition that the internal product by itself can be regarded as a similarity metric. We also define W slightly differently, ommiting the $X^\intercal$ factor as $W = (S + \alpha I)^{-1}Y$, so that the final prediction is given by

$$\hat{Y} = SW = (S + \alpha I)^{-1}S \tag{A.4}$$

For the bipartite interaction prediction case, besides standard adaptations as described in Section 2.2, a unique formulation is presented by 30. Similar in concept to the standard global single output procedure (Section 2.2), the authors consider each interaction pair as a unitary instance. The authors then propose building a kernel matrix relating each pair of instances to another pair, and not each interacting entity to another of the same domain. If $S_1 \in \mathbb{R}^{n_1 \times n_1}$ and $S_2 \in \mathbb{R}^{n_2 \times n_2}$ are the intra-domain similarity matrices, the global kernel matrix S is defined as

$$S^{[(i_1 n_2 + i_2)(j_1 n_2 + j_2)]} = S_1^{[i_1 j_1]} S_2^{[i_2 j_2]} \tag{A.5}$$

or, more succinctly, as the Kronecker product of S_1 and S_2:

$$S = S_1 \otimes S_2 \tag{A.6}$$

Each entry on S thus represents the similarity between the pair $X_1^{[i_1]}$-$X_2^{[j_1]}$ and another pair $X_1^{[i_2]}$-$X_2^{[j_2]}$ by the product of the similarities between $X_1^{[i_1]}$ and $X_1^{[j_1]}$ and between $X_2^{[i_2]}$ and $X_2^{[j_2]}$.

The bipartite linear regression is then framed on the vectorized Y, denoted vec(Y), built by concatenating the columns of Y into a single $|Y|$ by 1 column vector. Purely for notation purposes, we organize the weight parameters as the vectorized version of a matrix W with the same dimensions of Y.

$$\text{vec}(\hat{Y}) = S\,\text{vec}(W) \approx \text{vec}(Y) \tag{A.7}$$

$$\text{vec}(W) = (S + \alpha I)^{-1}\text{vec}(Y) \tag{A.8}$$

As the reader may imagine, S gets prohibitively large for big datasets (it's a $n_1 n_2$-sized square matrix!), both in terms of memory usage and the time needed to perform the matrix inversion. The authors, however, provide a clever way of circumventing this issue by decomposing each of the S_1 and S_2 kernel matrices separately and exploiting the properties of the Kronecker product.

Given that S_1 and S_2 are symmetric square matrices, it follows from the spectral theorem that they can be decomposed as follows:

$$S_1 = U_1 \Lambda_1 U_1^\intercal$$
$$S_2 = U_2 \Lambda_2 U_2^\intercal$$

where, if λ_1 represents the vector of eigenvalues of S_1, Λ_1 is the diagonal matrix of those eigenvalues ($\Lambda_1 = \text{diag}(\lambda_1)$), with U_1 columns representing their corresponding eigenvectors $U^{\intercal[i]}$ for each $\lambda_1^{[i]}$. The symmetry of the similarity matrices also implies that U_1 and U_2 are orthogonal, i.e. $U_1^\intercal U_1 = U_1 U_1^\intercal = \mathbb{I}$ and $U_2^\intercal U_2 = U_2 U_2^\intercal = \mathbb{I}$, or, equivalently, $U_1^{-1} = U_1^\intercal$ and $U_2^{-1} = U_2^\intercal$. Utilizing the fact that $(AB) \otimes (CD) = (A \otimes C)(B \otimes D)$ (108), the Kronecker product of S_1 and S_2 can be written as

$$S = S_1 \otimes S_2 = (U_1 \Lambda_1 U_1^\intercal) \otimes (U_2 \Lambda_2 U_2^\intercal) = (U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(U_1 \otimes U_2)^\intercal = U \Lambda U^\intercal \tag{A.9}$$

in which we denote $U = U_1 \otimes U_2$ and $\Lambda = \Lambda_1 \otimes \Lambda_2$. W now becomes

$$\text{vec}(W) = (U\Lambda U^\intercal + \alpha\mathbb{I})^{-1}\text{vec}(Y)$$

Further exploring the orthogonality of U, $UU^\intercal = U\mathbb{I}U^\intercal = \mathbb{I}$, so that

$$\text{vec}(W) = U(\Lambda + \alpha\mathbb{I})^{-1}U^\intercal\text{vec}(Y)$$

The most crucial property of the Kronecker product for our application is its relationship with the vectorization operator (108): $(A \otimes B)\text{vec}(C) = \text{vec}(BCA^\intercal)$, which allows us to write

$$\text{vec}(W) = U(\Lambda + \alpha\mathbb{I})^{-1}(U_1^\intercal \otimes U_2^\intercal)\text{vec}(Y) = U(\Lambda + \alpha\mathbb{I})^{-1}\text{vec}(U_2^\intercal Y U_1)$$

Since $(\Lambda + \alpha\mathbb{I})^{-1}$ is diagonal, its multiplication by the vector $\text{vec}(U_2^\intercal Y U_1)^\intercal$ can be expressed as a Hadamard product (element-wise multiplication, denoted by $\odot$) between two vectors. Acting element-wise, the Hadamard product is unaffected by vectorization, so that we can simplify the above expression by employing the matrix

$$(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1 [ij]} = \frac{1}{\alpha + \lambda_1^{[i]}\lambda_2^{[j]}} \tag{A.10}$$

In this context, $\lambda_1 \otimes \lambda_2^\intercal$ represents the n_1 by n_2 matrix resulting from the *outer product* of the vectors containing the eigenvalues of S_1 and S_2, respectively, while $A^{\circ -1}$ denotes the Hadamard inverse, corresponding to the matrix formed by taking the reciprocal of each element in A. Thus,

$$\begin{aligned}\text{vec}(W) &= U\text{diag}(\Lambda + \alpha^{-1}\mathbb{I}) \odot \text{vec}(U_2^\intercal Y U_1) = \\ &= (U_1 \otimes U_2)\text{vec}((\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)) = \\ &= \text{vec}(U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal)\end{aligned}$$

Which yields

$$W = U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal \tag{A.11}$$

Finally, predictions for a new group of instances in the test set are obtained as follows from the similarities with the train instances ($S_{1,\,\text{test}}^{[ij]}$ specifies the similarity between $X_{1,\,\text{test}}^{[i]}$ and $X_{1,\,\text{train}}^{[j]}$).

$$\text{vec}(\hat{Y}_{\text{test}}) = (S_{1,\,\text{test}} \otimes S_{2,\,\text{test}})\text{vec}(W) = \text{vec}(S_{2,\,\text{test}} W S_{1,\,\text{test}}^\intercal) \tag{A.12}$$

which summarizes to

$$\hat{Y}_{\text{test}} = S_{2,\,\text{test}} U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal S_{1,\,\text{test}}^\intercal \tag{A.13}$$

A.2 Neighborhood-Regularized Matrix Factorization

This section describes the NRLMF (31) algorithm, which is a matrix factorization technique that incorporates similarity information between samples in the learning process. It enables latent features of new instances to be inferred from the features of their neighbors. The method is an extension of the Logistic Matrix Factorization (LMF) algorithm, which is designed for recommendation tasks (26). NRLMF was successfully applied to diverse bipartite interaction prediction problems, such as drug-target (31), long non-coding RNA-protein (60), and lncRNA-microRNA (109).

A.2.1 Traditional matrix factorization

The idea behind matrix factorization is to find an approximation of the interaction matrix Y by decomposing it into two matrices U and V of lower dimensions, such that

$$\hat{Y} = UV^T \approx Y \tag{A.14}$$

The rows of U and V can be seen as learned representations in a new vector space (usually called *latent* space) of each sample in the row and column domains, so that $U^{[i]}$ represents $X_1^{[i]}$ and $V^{[j]}$ represents $X_2^{[j]}$. Notice that the number of latent features is constrained by Equation A.14 to be the same for both instance domains: $|U|_j = |V|_j$, being an arbitrary hyperparameter to be defined by the user. Usually, the number of latent features is set to be much smaller than the number of original features ($|U|_j \ll |X_1|_j$ and $|V|_j \ll |X_2|_j$), requiring less computational labor and generating models less susceptible to overfitting.

The learning procedure of matrix factorization algorithms thus consists of obtaining such latent feature matrices U and V so to approximate Y as best as possible. The most common approach is to define a loss function that penalizes the difference between the predicted and the true values of Y and to employ gradient descent techniques to gradually change U and V in the direction that minimizes such loss.

As can be deduced from Equation A.14, the dot product of each row of U and V approximates the corresponding element of Y:

$$\hat{Y}^{[ij]} = U^{[i]} \cdot V^{[j]} \tag{A.15}$$

Logistic matrix factorization (LMF) slightly redefines the problem by introducing one more step to obtain $\hat{Y}$ from U and V (26): it assumes that the interaction matrix Y is the result of the logistic function applied to $UV^\intercal$ and not only $UV^\intercal$ anymore.

$$\hat{Y}^{[ij]} = \text{logistic}(U^{[i]} \cdot V^{[j]}) = \frac{\exp(U^{[i]} \cdot V^{[j]})}{1 + \exp(U^{[i]} \cdot V^{[j]})} \tag{A.16}$$

where $\mathbf{a} \cdot \mathbf{b}$ represents the dot product between the vectors $\mathbf{a}$ and $\mathbf{b}$. When applied to a matrix, we assume that $\log$ and $\exp$ functions operate element-wise:

$$(\log M)^{[ij]} = \log M^{[ij]}$$

If $\hat{Y}$ is interpreted as the probability of being positive as predicted by the model ($\hat{Y}^{[ij]} = P(Y^{[ij]} = 1)$ and $1 - \hat{Y}^{[ij]} = P(Y^{[ij]} = 0)$), the optimization objective is usually based on maximizing the joint probability of correctly guessing all interactions in Y:

$$P(\hat{Y} = Y) = \prod_{ij} |\hat{Y}^{[ij]} + Y^{[ij]} - 1| \tag{A.17}$$

in which $|a|$ represents the absolute value of a. A few modifications are then further made:

1. The logarithm of the objective function is taken to simplify the expression without affecting the optimization problem, since $\log$ is a monotonic function;
2. Since positive interactions are usually less numerous but more important in matrix completion problems, a factor α is introduced to prioritize them, multiplying the terms corresponding to $Y^{[ij]} = 1$ in the objective function. It results as if $alpha$ copies of each positive interaction are present in the training set;
3. To discourage overfitting and avoid U and V being arbitrarily large, quadratic regularization terms are added, penalizing the magnitude of the elements of U and V.
4. Similarity information between samples is incorporated by NRLMF,

To simplify notation, we define matrices J whose combined sum of all elements corresponds to the objective function $\mathcal{J}$:

$$\mathcal{J} = \sum J_{\text{labels}} + \sum J_{1,\text{ reg.}} + \sum J_{2,\text{ reg.}} + \sum J_{1,\text{ neigh.}} + \sum J_{2,\text{ neigh.}} \tag{A.18}$$

where by $\sum J$ we denote $\sum_i^{|J|_i} \sum_j^{|J|_j} J$. The sums must be performed individually due to the J matrices having different dimensions.

Applying the logarithm to Equation A.17 we have our first term of $\mathcal{J}$:

$$J_{\text{labels}} = \log|\hat{Y} + Y - 1| = Y \odot \log \hat{Y} + (1 - Y) \odot \log(1 - \hat{Y})$$

where we separate the cases in which $Y^{[ij]} = 1$ and $Y^{[ij]} = 0$. $\odot$ represents the Hadamard product (element-wise multiplication) between matrices:

$$(A \odot B)^{[ij]} = A^{[ij]} B^{[ij]}$$

Adding the aforementioned positive importance factor α and expanding $\hat{Y}$ according to Equation A.16 we have

$$\begin{aligned} J_{\text{labels}} = \alpha Y \odot \{UV^\intercal - \log[1 + \exp(UV^\intercal)]\} - (1 - Y) \odot \log[1 + \exp(UV^\intercal)] = \\ = \alpha Y \odot UV^\intercal + [(1 - \alpha)Y - 1] \odot \log[1 + \exp(UV^\intercal)] \end{aligned} \tag{A.19}$$

To discourage large values in U and V, we consider quadratic regularization terms weighted by hyperparameters λ_1 and λ_2:

$$J_{1,\text{ reg.}} = -\frac{\lambda_1}{2} U \odot U = -\frac{\lambda_1}{2} \mathbb{I} \odot (UU^\intercal) \tag{A.20}$$

$$J_{2,\text{ reg.}} = -\frac{\lambda_2}{2} V \odot V = -\frac{\lambda_2}{2} \mathbb{I} \odot (VV^\intercal) \tag{A.21}$$

If the initial objective proposal can be interpreted as maximizing the probability of guessing all labels correctly given specific U and V ($\sum J_{\text{labels}} = \log P(\hat{Y} = Y \mid U, V)$), adding the regularization terms is equivalent to introduce prior assumptions about U and V distributions and define a slightly different objective: maximizing the posterior probability to obtain the current U and V given that $\hat{Y} = Y$. Applying Bayes' theorem and assuming $P(Y) = 1$ we have

$$P(U, V \mid \lambda_1, \lambda_2, Y) = P(Y \mid U, V)\; P(U \mid \lambda_1)\; P(V \mid \lambda_2) \tag{A.22}$$

Under the assumption that the values in U and V follow zero-centered spherical gaussian distributions with variances given by $\frac{1}{\lambda}$, that is, $U^{[ij]} \sim \mathcal{N}(0, \lambda_1^{-1}\mathbf{I})$ and $V^{[ij]} \sim \mathcal{N}(0, \lambda_2^{-1}\mathbf{I})$, we recover the regularized objective function of Equation A.21 (26).

$$\begin{aligned}
&\log P(U, V \mid \lambda_1, \lambda_2, Y) = \\
&\quad = \log P(Y \mid U, V) + \sum \log(\exp(-\frac{\lambda_1}{2} U \odot U)) + \sum \log(\exp(-\frac{\lambda_2}{2} V \odot V)) = \\
&\quad\quad = \sum J_{\text{labels}} - \sum \frac{\lambda_1}{2} U \odot U - \sum \frac{\lambda_2}{2} V \odot V
\end{aligned} \tag{A.23}$$

Therefore, if multiple values of U and V possibly generate the same $\hat{Y} = Y$, applying the regularization can be understood as not only finding one of such combinations but, between those U and V that satisfy $\hat{Y} = Y$, finding the U and V that are most likely to be randomly sampled. If $P(U, V \mid \lambda_1, \lambda_2, Y)$ continuously varies as a function of U and V, pooling U and V from a region of maximal $P(U, V \mid \lambda_1, \lambda_2, Y)$ should improve generality, arguably ensuring that stochastic deviations of U and V would still result in high $P(\hat{Y} = Y)$ and justifying the use of regularization as a way to avoid overfitting.

One may have noticed that the original feature matrices X_1 and X_2 were not considered in any regard when describing matrix factorization and detailing the objective functions. Born in the context of recommendation systems where the relationship labels are usually the only information available, matrix factorization algorithms in general encounter a significant issue when brought to our current scenario of bipartite interaction prediction: in its canonic formulation, they do not take sample-level features into account, often called *side information* or *side features* in the recommendation field (110), possibly overlooking valuable data. As a consequence, they are unable to provide predictions for new samples that were not present in the training set, since no information about them is available to be inputted to the model. This is commonly regarded as the *cold-start problem* (9). As such, matrix factorization usage is usually restricted to the task of matrix completion, in which the goal is to predict the missing values of a matrix given the values of the remaining elements, without receiving completely new rows or columns during model evaluation (9).

A.2.2 Neighborhood regularization

Targeting these issues, 31 proposes a modification to LMF that incorporates side information into the model, enabling predicting interactions for new samples. The core idea of their technique lies in adding one more term to the objective function, penalizing instances regarded as close when considering the original features but were separated by U and V, placed far from each other in the latent space. As such, the algorithm is called *Neighborhood-Regularized Logistic Matrix Factorization* (NRLMF). To precisely define it, let's consider similarity-weighted adjacency matrices A_1 and A_2 referring to each sample domain that specifies neighborhood relationships between samples. If $S_1^{[ij]}$ denotes a similarity score between $X_1^{[i]}$ and $X_1^{[j]}$, $A_1^{[ij]}$ is set to this similarity value if $X_1^{[j]}$ belongs to the neighborhood of $X_1^{[i]}$, denoted $N(X_1^{[i]})$, and 0 otherwise:

$$A_1^{[ij]} = \begin{cases} S_1^{[ij]} & \text{if } X_1^{[j]} \in N(X_1^{[i]}) \\ 0 & \text{otherwise} \end{cases} \tag{A.24}$$

Multiple options are available for the definition of neighborhoods, such as considering all samples within a certain radius of each other or only the k nearest neighbors of each sample. In this work, following the original proposal of NRLMF (31), we will consider the latter, defining A_1 and A_2 as the adjacency matrices of the k-nearest neighbors graphs of X_1 and X_2, respectively. In other words, $N(X_1^{[i]})$ is the set formed by the k rows $X_1^{[j]}$ with the k highest s_ij. In the interaction prediction problems we analyze, similarities are precalculated so that the X matrices directly provide the distance metric over which the nearest neighbors are selected. That is, X_1 and X_2 themselves already constitute pairwise similarity matrices: $S_1^{[ij]} = X_1^{[ij]}$. In general, however, one may need to define a kernel matrix S as a preprocessing step, choosing a distance metric over the original features to be used in the nearest neighbors search such as the Euclidean distance or a radial basis function (30). In any case, notice that A is a function of X alone for NRLMF. While Y may also be considered in similar scenarios (as will be discussed ahead), A does not depend on U or V and can be built as a single pre-training step.

The loss term proposed by NRLMF is then given by the sum of the Euclidean distances in the latent space between samples in the same neighborhood, weighted by their similarities:

$$\mathcal{J}_{1,\text{ neigh.}} = -\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 \tag{A.25}$$

$$\mathcal{J}_{2,\text{ neigh.}} = -\sum_{ij} A_2^{[ij]} \|V^{[i]} - V^{[j]}\|^2 \tag{A.26}$$

in which $\|\mathbf{v}\|$ represents the Euclidean norm of a vector $\mathbf{v}$. Concentrating on the row instances and expanding the last definition we have

$$\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 = \sum_{ij} A^{[ij]} \left(U^{[i]} \cdot U^{[i]} + U^{[j]} \cdot U^{[j]} - 2U^{[i]} \cdot U^{[j]}\right) =$$

$$= \sum_i \left(\sum_j A_1^{[ij]}\right) U^{[i]} \cdot U^{[i]} + \sum_j \left(\sum_i A_1^{[ij]}\right) U^{[j]} \cdot U^{[i]}$$

The terms in which U appears with the same index ($U^{[i]} \cdot U^{[i]}$ and $U^{[j]} \cdot U^{[j]}$) can be rewritten to include both by multiplying them by the identity matrix $\mathbb{I}$. Essentially, we consider $\sum_i U^{[i]} \cdot$

$U^{[i]} = \text{trace}(UU^\intercal) = \sum_{ij}(\mathbb{I} \odot UU^\intercal)^{[ij]}$.

$$\sum_i \left(\sum_j A_1^{[ij]}\right) U^{[i]} \cdot U^{[i]} = \sum_i \left(\sum_j A_1^{[ij]}\right) \sum_k \mathbb{I}^{[ik]} U^{[i]} \cdot U^{[k]} = \sum_{ij} \left(\sum_k A_1^{[ik]}\right) \mathbb{I}^{[ij]} U^{[i]} \cdot U^{[j]}$$

This allows us to write

$$\begin{aligned} &\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 = \\ &\qquad = \sum_{ij} \left[\left(\sum_k A_1^{[ik]} + \sum_l A_1^{[lj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} \right] U^{[i]} \cdot U^{[j]} = \\ &\qquad\qquad = \sum L_1 \odot (UU^\intercal) \end{aligned}$$

in which we define

$$L_1^{[ij]} = \left(\sum_k A_1^{[ik]} + \sum_l A_1^{[lj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} = \left(\sum_k A_1^{[ik]} + A_1^{[kj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} \tag{A.27}$$

Notice that taking $A_1^\intercal$ instead of A_1 has no effect on the final result, since $\sum A_1 UU^\intercal = \sum A_1^\intercal UU^\intercal$. We could then work with a symmetrized version of A_1 from the start:

$$\tilde{A}_1 = A_1 + A_1^\intercal$$

yielding

$$L_1 = \left(\sum_k \tilde{A}_1^{[ik]} \right) \mathbb{I}^{[ij]} - \tilde{A}_1^{[ij]}$$

We can see that the first term of L_1 acts in a similar way to the quadratic regularization terms presented by Equation A.21, multiplying the main diagonal of $UU^\intercal$ and thus penalizing the model for latent vectors with large Euclidean norms (the diagonal of $UU^\intercal$ holds the squared norms $U^{[i]} \cdot U^{[i]}$). The amount of regularization is however pondered by the weighted number of neighbors of each sample in this case: $\sum_k A_1^{[ik]}$ represents the sum of similarities of $X_1^{[i]}$ with all its neighbors, also called the *degree* of a sample. This results in samples with larger and more compact neighborhoods being more heavily penalized for having large norms in the latent space. The second term of Equation A.27, on the other hand, rewards the model for placing close neighbors colinear to each other in the latent space, summing over $S^{[ij]} U^{[i]} \cdot U^{[j]}$ terms between each sample and its neighbors ($A_1^{[ij]}$ is 0 if $U^{[i]}$ and $U^{[j]}$ are not neighbors).

Finally, the neighborhood regularization terms are written as

$$J_{1,\text{ neigh.}} = -\frac{\beta_1}{2} L_1 \odot (UU^\intercal) \tag{A.28}$$

$$J_{2,\text{ neigh.}} = -\frac{\beta_2}{2} L_2 \odot (VV^\intercal) \tag{A.29}$$

Combining the matrix terms as in Equation A.18, NRLMF's objective function is given by

$$\begin{aligned}\mathcal{J} = &\sum \alpha Y \odot UV^\intercal + [(1-\alpha)Y - 1] \odot \log\left[1 + \exp(UV^\intercal)\right] \\ &- \sum \frac{1}{2}(\lambda_1 \mathbb{I} + \beta_1 L_1) \odot (UU^\intercal) \\ &- \sum \frac{1}{2}(\lambda_2 \mathbb{I} + \beta_2 L_2) \odot (VV^\intercal)\end{aligned} \tag{A.30}$$

and the derivatives of the objective function with respect to U and V to be used in the gradient descent procedure are given by

$$G_U = \frac{\partial \mathcal{J}}{\partial U} = \{[(1-\alpha)Y - 1] \odot \hat{Y} + \alpha Y\}V - (\lambda_1 \mathbb{I} + \beta_1 L_1)U \tag{A.31}$$

$$G_V = \frac{\partial \mathcal{J}}{\partial V} = \{[(1-\alpha)Y - 1] \odot \hat{Y} + \alpha Y\}^\intercal U - (\lambda_2 \mathbb{I} + \beta_2 L_2)V \tag{A.32}$$

The training procedure of NRLMF is presented by Function TrainNRLMF, and consists of alternated updates on U and V in the gradient's direction until certain stop criteria are satisfied. Common choices for stopping conditions are a maximum number of iterations or a minimum change in the objective function between iterations.

Since faster convergence is reported by the original authors (26), we follow previous work (26, 31, 64, 65) by implementing the AdaGrad procedure (111), in which the length of each gradient step is divided by the square-root sum of squared previous steps:

$$U_{t+1} = U_t + \frac{\eta G_{U,t}}{\sqrt{\sum_{t'=0}^{t} G_{U,t'}^2}} \tag{A.33}$$

where $G_{U,t}$ is the partial derivative $\frac{\partial \mathcal{J}}{\partial U}$ of the objective function with respect to U at step t, and η is the user-defined learning rate. The same is done for V.

The main importance of NRLMF however lies in the inference phase. As mentioned, matrix factorization methods are not designed to deal with new input samples, that are not present in the training set. Specifically, traditional matrix factorization is incapable of generating latent vectors for the unseen samples to be used for label prediction. An idea that may seem natural at first glance is to delay training to the arrival of new instances, including them in the training set with zero-only labels before performing the optimization. But even then, using an objective function based only on Y as is traditionally done, no new information is brought by the new instances and adding new zeroed rows or columns to Y will mainly introduce noise to the training data and likely degrade the model's predictive performance.

NRLMF, however, leverages proximity information encoded by X to remarkably enable determining latent feature vectors for completely new instances. The neighborhood regularization terms in the objective function now reveal their full importance: they support proximity as a transferable property between the original and the latent spaces. By encouraging that neighbors

```
Function TrainNRLMF($Y$, $S_1$, $S_2$, $\alpha$, $\lambda_1$, $\lambda_2$, $\beta_1$, $\beta_2$, $\eta$): Train an NRLMF model.
    Input: $Y$, the training labels matrix to be approximated.
    Input: $S_1$, $S_2$, Similarity matrices among instances of each axis.
    Input: $\alpha$, the positive importance factor.
    Input: $\lambda_1$, $\lambda_2$, quadratic regularization factors.
    Input: $\beta_1$, $\beta_2$, neighborhood regularization factors.
    Input: $\eta$, the learning rate.
    Output: $U$, $V$, the resulting latent feature matrices.
    Optionally precompute constant factors of the gradient (Equations A.31 and A.32), such as $(\lambda_2\mathbb{I} + \beta_2 L_2)$, $[(1-\alpha)Y - 1]$ or $\alpha Y$;
    Initialize $U$ and $V$ with normally-distributed random values;
    $T_U, T_V \leftarrow \mathbf{0}$;            // Initialize gradient accumulators
    while Stop conditions are not met do
        // Update U
        Obtain $G_U$ through Equation A.31;
        $T_U \leftarrow T_U + G_U^2$;
        $U \leftarrow U + \eta \frac{G_U}{\sqrt{T_U}}$;
        // Update V
        Obtain $G_V$ through Equation A.32;
        $T_V \leftarrow T_V + G_V^2$;
        $V \leftarrow V + \eta \frac{G_V}{\sqrt{T_V}}$;
    end
    return $U$, $V$
```

in X_1 and X_2 remain close in U and V, we can infer latent features of new instances based on their neighborhood.

Consider the test similarity matrices $S_{1,\,\text{test}}$ and $S_{2,\,\text{test}}$ respectively derived from $X_{1,\,\text{test}}$ and $X_{2,\,\text{test}}$, relating the new instances to the known training samples. For instance, $S_{1,\,\text{test}}^{[ij]}$ represents the similarity between $X_{1,\,\text{test}}^{[i]}$ and $X_{1,\,\text{train}}^{[j]}$. If $A_{1,\,\text{test}}$, like before in Equation A.24, accordingly restricts the similarity matrix to the neighborhood of each sample,

$$A_{1,\,\text{test}}^{[ij]} = \begin{cases} S_{1,\,\text{test}}^{[ij]} & \text{if } X_{1,\,\text{train}}^{[j]} \in N(X_{1,\,\text{test}}^{[i]}) \\ 0 & \text{otherwise} \end{cases} \tag{A.34}$$

the latent feature vector of a new instance is simply estimated as the weighted average of its neighbors' latent representations:

$$U_{\text{test}}^{[i]} = \frac{A_{1,\,\text{test}}^{[i]} U_{\text{train}}}{\sum A_{1,\,\text{test}}^{[i]}} \tag{A.35}$$

the analogous being held for V, so that new predictions are made as usual with Equation A.16, where U_{test}, U_{train}, V_{test} and V_{train} can be used in accordance with the prediction task under study (see Section 2.7.2 for details on the different prediction scenarios):

$$\hat{Y}_{\mathrm{TT}} = \frac{\exp(U_{\text{test}} V_{\text{test}}^{\intercal})}{1 + \exp(U_{\text{test}} V_{\text{test}}^{\intercal})} \quad \hat{Y}_{\mathrm{TL}} = \frac{\exp(U_{\text{test}} V_{\text{train}}^{\intercal})}{1 + \exp(U_{\text{test}} V_{\text{train}}^{\intercal})} \quad \hat{Y}_{\mathrm{LT}} = \frac{\exp(U_{\text{train}} V_{\text{test}}^{\intercal})}{1 + \exp(U_{\text{train}} V_{\text{test}}^{\intercal})} \tag{A.36}$$